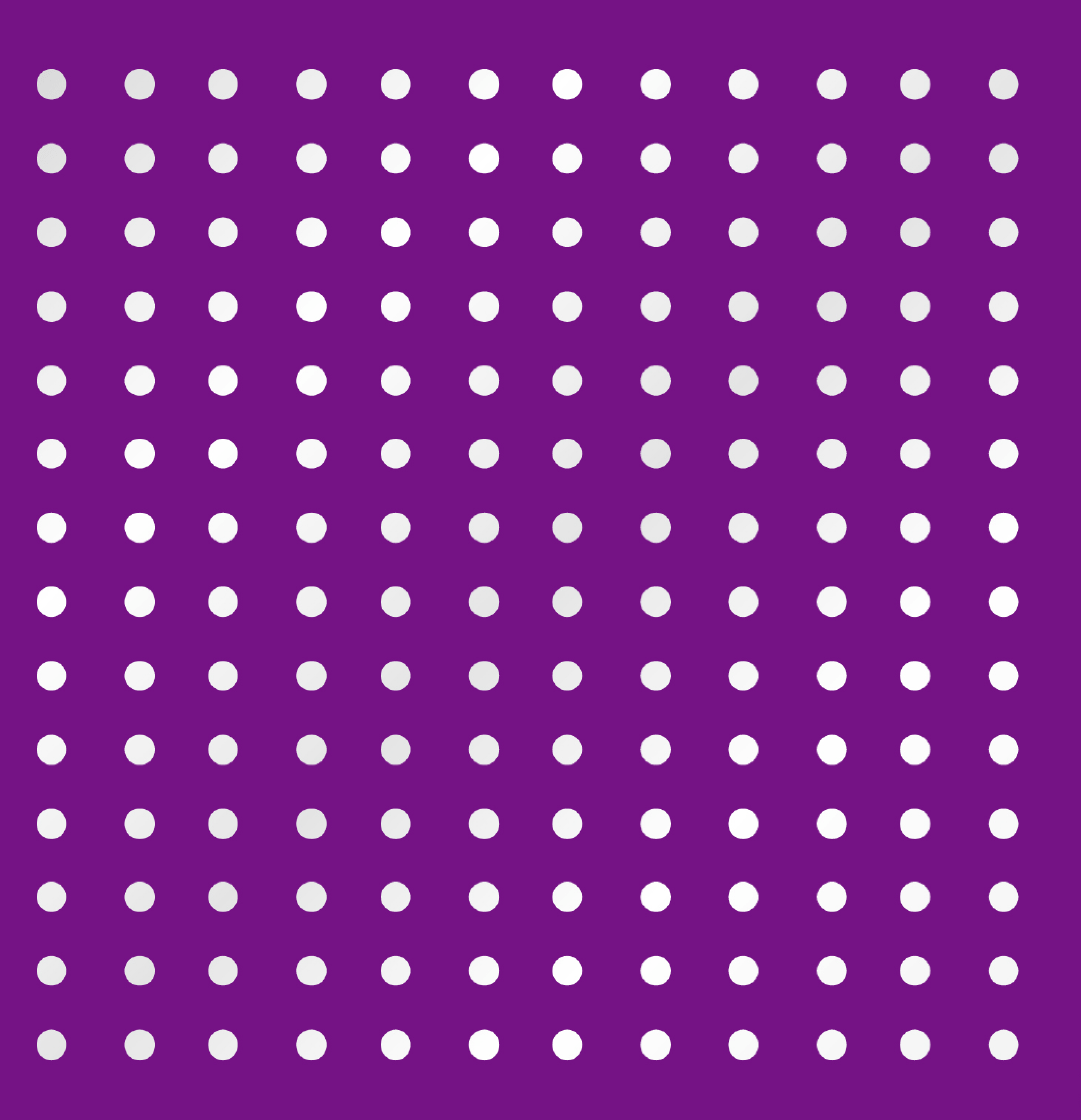
高等院校信息技术规划教材

# Android 应用程序开发

王向辉 张国印 沈洁 编著



清华大学出版社

高等院校信息技术规划教材

# Android 应用程序开发

王向辉　张国印　沈　洁　编著

清华大学出版社

北　京

## 内容简介

Android是一个优秀的开源手机平台。本书由浅入深地介绍了Andriod应用程序的开发，内容共分11章，包括Android的简介，开发环境，应用程序、Android生命周期和用户界面，组件通信与广播消息，后台服务，数据存储与访问，位置服务与地图应用，Android NDK开发以及综合示例设计与开发。

本书内容丰富，实用性强，既可用做高等院校信息技术的教材，也可供相关专业人士参考。



**图书在版编目（CIP）数据**

Android应用程序开发/王向辉，张国印，沈洁编著．—北京：清华大学出版社，2010.3
（高等院校信息技术规划教材）
ISBN 978-7-302-21699-5

Ⅰ．①A…　Ⅱ．①王…　②张…　③沈…　Ⅲ．①移动通信－携带电话机－应用程序－程序设计－高等学校－教材　Ⅳ．①TN929.53

中国版本图书馆CIP数据核字(2009)第243018号

**责任编辑：**袁勤勇　林晴嘉
**责任校对：**焦丽丽
**责任印制：**王秀菊

**出版发行：**清华大学出版社　　**地　　址：**北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编：**100084
**社　总　机：**010-62770175　　**邮　　购：**010-62786544
**投稿与读者服务：**010-62776969，c-service@tup.tsinghua.edu.cn
**质　量　反　馈：**010-62772015，zhiliang@tup.tsinghua.edu.cn
**印　装　者：**北京嘉实印刷有限公司
**经　　销：**全国新华书店
**开　　本：**185×260　　**印　　张：**17.25　　**字　　数：**396千字
**版　　次：**2010年3月第1版　　**印　　次：**2010年3月第1次印刷
**印　　数：**1～3000
**定　　价：**28.00元

---

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：035504-01

# 前言 Foreword

Android 是谷歌(Google)发布的一个开放源代码的手机平台,由 Linux 内核、中间件、应用程序框架和应用软件组成,是第一个可以完全定制、免费、开放的手机平台。Android 不仅能够在智能手机中使用,还可以用在移动互联网终端(MID)、上网笔记本、便携式媒体播放器(PMP)和汽车电子等其他手持设备上。

Android 在诞生之日起便受到广泛的关注。到目前为止,中国台湾的宏达电(HTC)已推出了三款 Android 手机,LG、三星、摩托罗拉、索尼爱立信、宏碁、华硕和联想也相继推出自己的 Android 手机,中国移动、中国联通等运营商也在研发基于该系统的手机操作平台。据市场调查机构预期,2009 年全球 Android 手机的总规模达 800 万部,2013 年全球应用 Android 操作系统的产品销售量将超过 4000 万台。

本书基于 Android SDK 的 1.5 版本,全面而详细地介绍了 Android 应用程序开发所涉及的各个方面内容,包括集成开发环境的搭建,用户界面和关键组件的使用方法,地图应用和 Android NDK 开发等内容。从外到内、由表及里地介绍了 Android 系统的各种特性,将 Android 系统的优越之处展现在读者的面前,通过每章的内容逐渐引领读者进入 Android 的世界。

全书的内容包括:

第 1 章介绍了 Android 平台的起源、发展、特征和体系结构,对比分析了 Windows Mobile、PalmOS、Symbian、Linux 和 iPhoneOS 等主流手机系统的优势和不足。

第 2 章详细说明了 Android 开发环境的安装与配置方法,并对开发和调试过程中可能使用到的工具进行了简单的介绍。

第 3 章介绍了基于 Eclipse 开发 Android 应用程序的基础知识和基本方法,说明了 Android 工程文件的结构和用途,并介绍了使用命令行开发、安装和运行 Android 应用程序的方法。

第 4 章介绍了 Android 程序的生命周期和进程优先级的变更方式,并以 Activity 为例说明 Android 组件生命周期的状态转换和事

件回调函数的调用顺序，最后简单介绍了 Android 调试工具的使用方法。

第 5 章介绍了 Android 用户界面的开发方法，重点介绍了常见的界面控件，界面布局，菜单和界面事件的使用方法。

第 6 章介绍了 Android 系统的组件通信机制，其中包括使用 Intent 启动组件的原理和方法，Intent 过滤器的原理与匹配机制，广播消息的接收和发送方法等。

第 7 章介绍了 Android 系统的后台服务组件 Service，内容包括 Service 的原理和用途，Service 的启动和绑定，AIDL 语言定义跨进程服务的接口，以及线程使用和跨线程界面更新。

第 8 章介绍了 Android 系统所提供的多种数据存储方法，其中包括易于使用的 SharedPreferences、经典的文件存储和轻量级的 SQLite 数据库，最后介绍了 Android 系统的应用程序间数据共享接口 ContentProvider。

第 9 章介绍了位置服务的概念和位置信息获取方法，简单说明了 Google 地图密钥的申请方法，重点介绍了 Google 地图中的 MapView、MapController 和 Overlay 的使用方法。

第 10 章介绍了 Android 系统中使用 C/C++ 等非托管代码开发应用程序的方法，其中包括 Android NDK 的用途和优缺点，编译环境的安装与配置方法，以及 Android NDK 的开发方法。

第 11 章以“天气预报短信服务软件”为例，介绍了 Android 应用程序开发过程中需求分析、界面设计、模块设计和程序开发等步骤，并简单介绍了 Android 应用程序的设计和开发的思路与方法。

本书由哈尔滨工程大学计算机科学与技术学院王向辉、张国印和沈洁担任主要编写工作。同时，参与本书编写和校对的人员还有翁岩青、杜婧、徐子涵、邹新和马书亮，这里对他们的辛苦工作表示衷心的感谢。在本书的编写过程中，得到黑龙江省电子信息产品监督检验院王希忠院长的热情帮助，并获得黑龙江省金源数码科技发展有限公司和黑龙江省网络与软件测评中心的技术支持，在此表示感谢。

Android 是一个新兴的手机平台，各个方面还在不断地发展和变化。由于能力和水平所限，虽然竭尽全力，但仍然难免存在错误和疏漏的地方，希望各位专家、读者能毫不保留地提出所发现的问题，与编者共同讨论，联系方式是 wangxianghui@hrbeu. edu. cn。

**编　者**

**2009 年 10 月于哈尔滨工程大学**

# 目录 Contents

# 第1章 chapter 1

# Android 简 介

Android是一个优秀的开源手机平台,通过本章的学习可以让读者对Android平台的起源、发展、特征和体系结构有个初步的了解,然后简要介绍了Windows Mobile、PalmOS、Symbian、Linux和iPhoneOS等目前主流的手机操作系统,了解各个主流手机操作系统的优势和不足。

**本章学习目标:**

- 了解各种手机操作系统的特点
- 了解开放手机联盟的目的、性质和组成
- 了解Android平台的发展历史
- 掌握Android平台的特征
- 掌握Android平台的体系结构

## 1.1 手机操作系统

在早期的手机内部是没有操作系统的,所有的软件都是由手机生产商在设计时所定制的,因此手机在设计完成后基本是没有扩展功能的。后期的手机为了提高手机的可扩展性,使用了专为移动设备开发的操作系统,使用者可以根据需要安装不同类型的软件。虽然使用操作系统的手机具有更好的可扩展性,但由于操作系统对于手机的硬件配置要求较高,所产生的硬件成本和操作系统成本使手机的售价明显高于不使用操作系统的手机,因此一般只有在高端智能手机上使用手机操作系统。

目前应用在手机上的操作系统主要有6种,分别是Windows Mobile、PalmOS、Symbian、Linux、iPhoneOS和Android。

Windows Mobile是微软推出的移动设备操作系统,捆绑了一系列针对移动设备而开发的应用软件,这些软件构建在Microsoft Win32 API基础之上,可以播放音视频文件、浏览网页、MSN聊天和收发电子邮件。由于该操作系统对硬件配置要求较高,一般需要使用高主频的嵌入式处理器,从而产生了耗电量大、电池续航时间短和硬件成本高等缺点。Windows Mobile系列操作系统包括Smartphone、Pocket PC和Portable Media Center。Smartphone提供的功能侧重点在联系方面,主要支持的功能有电话、电子邮件、联系人和即时消息等。Pocket PC的功能侧重个人事务处理和简单的娱乐,主要支持的

功能有日程安排、移动版 Office 和多媒体播放功能等等。Portable Media Center 提供的功能侧重点在移动多媒体功能,主要支持音频播放和视频播放等。图 1.1 是 Windows Mobile 的用户界面。

PalmOS 是 32 位的嵌入式操作系统,主要在移动终端上使用。PalmOS 由 3Com 公司的 Palm Computing 部开发,拥有较多的第三方软件。PalmOS 在设计时考虑到了移动设备的内存相对较小,所以操作系统本身所占的内存极小,基于 PalmOS 编写的应用程序所占的空间也很小。PalmOS 的操作界面采用触控式,基本所有的控制选项都排列在屏幕上,仅使用手写笔就可以完成所有操作。PalmOS 向用户免费提供了开发工具,允许用户利用该工具编写或修改相关软件,使支持 PalmOS 的应用程序丰富多彩。PalmOS 在其他方面还存在一些不足,例如自身不具有录音和 MP3 播放功能,如果需要使用这些功能,还需要加入第三方软件或硬件设备方可实现。图 1.2 是 PalmOS 的用户界面。

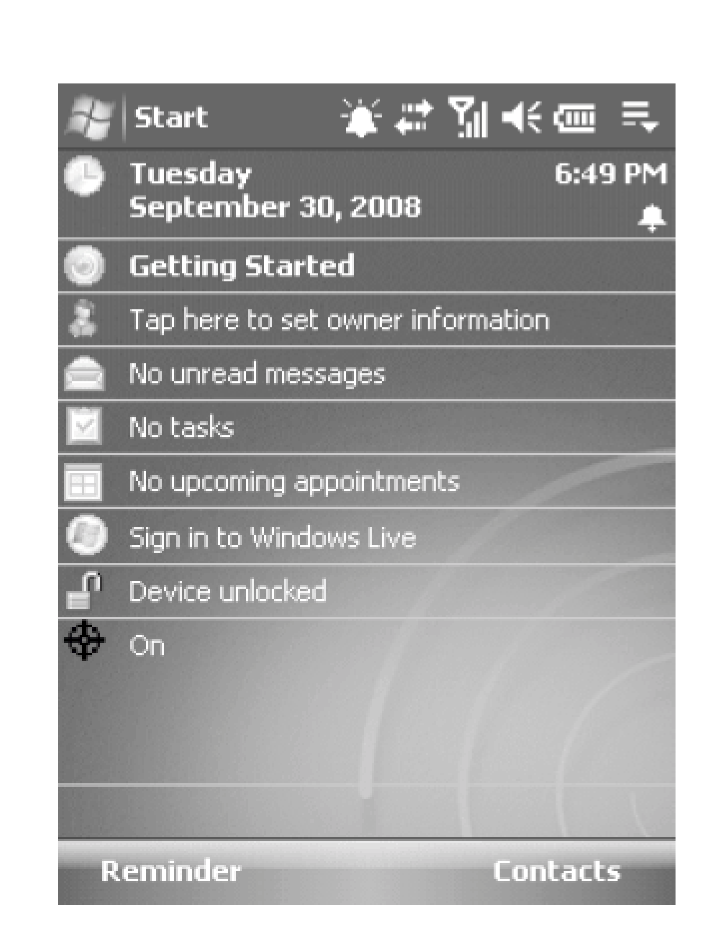



图 1.1　Windows Mobile 的用户界面




图 1.2　PalmOS 的用户界面

Symbian 是为手机而设计的实时多任务 32 位操作系统,提供了开发使用的函数库、用户界面、通用工具和参考示例。Symbian 操作系统由 Symbian 公司负责开发与维护,2008 年 12 月 2 日,诺基亚收购了 Symbian 公司。Symbian 操作系统具有功耗低、内存占用少等特点,适合手机等移动设备使用,而且有灵活的应用界面框架,不但使程序开发人员可以快速地掌握关键技术,还可以使手机制造商推出不同界面的产品。经过不断完善,Symbian 操作系统已经可以支持 3G、GPRS 和蓝牙等功能。Symbian 操作系统并不是完全开放的,它并没有开放核心代码,核心代码仅可以提供给使用 Symbian 操作系统的手机制造商和其他合作伙伴。但 Symbian 操作系统提供公开的 API 文档,任何人都可以开发用于 Symbian 操作系统的应用软件。图 1.3 是 Symbian 的用户界面。

Linux 手机操作系统是由计算机 Linux 操作系统演变而来的。在 Linux 成为移动终端的操作系统后,就以其开放源代码的优势吸引了越来越多的终端厂商和运营商的关注。Linux 开放源代码的特性不仅能够大幅度地降低手机的软件成本,还有利于独立软件开发商开发出硬件利用效率高、功能更强大的应用软件,行业用户开发出安全、可靠的应用系统;同时,开源也满足了手机制造商定制 Linux 手机操作系统的要求,吸引了众多软件开发商,丰富了第三方应用。然而,Linux 操作系统有其先天的不足。首先,入门难

度高，熟悉其开发环境的工程师少，而且集成开发环境较差；其次，由于微软操作系统源代码的不公开，基于 Linux 的产品与个人计算机的连接性较差；最后，尽管目前从事 Linux 操作系统开发的公司数量较多，但真正具有很强开发实力的公司却很少，而且这些公司之间是相互独立开发的，很难实现更大的技术突破。图 1.4 是 Linux 的用户界面。




图 1.3　Symbian 的用户界面




图 1.4　Linux 的用户界面

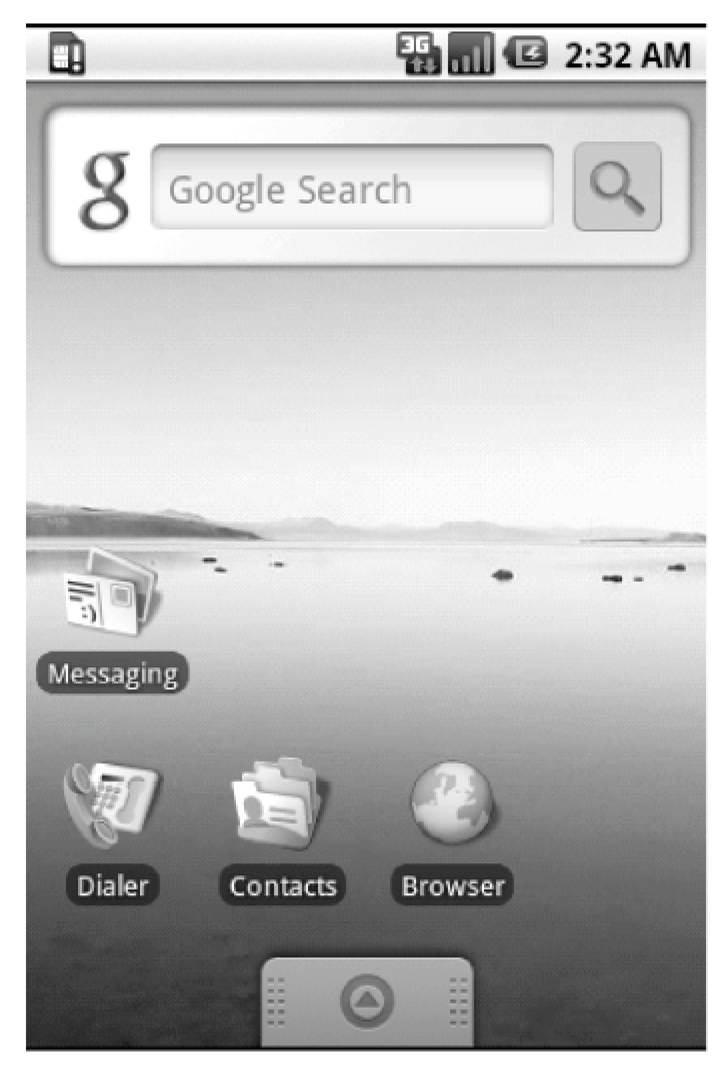
iPhoneOS 是由苹果公司为 iPhone 开发的操作系统，以开放源代码的操作系统 Darwin 为基础，主要是供苹果公司生产的 iPhone 手机和 MP4 播放器 iPod touch 使用。iPhoneOS 的系统架构分为 4 个层次，分别是核心操作系统层、核心服务层、媒体层和可轻触层。为了便于 iPhone 应用程序开发，苹果公司提供了 iPhone SDK，为 iPhone 应用程序进行开发、测试、运行和调试提供工具。多点触摸操作是 iPhoneOS 的用户界面基础，也是 iPhoneOS 区别与其他手机操作系统的特性之一，支持的控制方法包括滑动、轻按、挤压和旋转。此外，iPhoneOS 还通过支持内置加速器，允许系统界面根据屏幕的方向而改变方向。iPhoneOS 自带大量的应用程序，包括 SMS 简讯、日历、相机、YouTube、股市、地图、天气、时间、计算器、备忘录、系统设定、iTunes 和通讯录等。图 1.5 是 iPhoneOS 的用户界面。




图 1.5　iPhoneOS 的用户界面

Android 也是一种手机操作系统，是谷歌(Google)发布的基于 Linux 的开源手机平台，该平台由操作系统、中间件、用户界面和应用软件组成，是第一个可以完全定制、免费、开放的手机平台。区别于上述几个手机操作系统，Android 是一个完全免费的手机平台，使用 Android 并不需要授权费，而且因为 Android 平台有丰富的应用程序，也大幅度降低了应用程序的开发费用，可以节约 15%～20%的手机制造成本。Android 底层使用开源的 Linux 操作系统，同时开放了应用程序开发工具，使所有程序开发人员都在统一、开放的开发平台上进行开发，保证了 Android

应用程序的可移植性。Android 平台使用 Java 语言进行开发,支持 SQLite 数据库、2D/3D 图形加速、多媒体播放和摄像头等硬件设备,并内置了丰富的应用程序,如电子邮件客户端、闹钟、Web 浏览器、计时器、通讯录和 MP3 播放器等。图 1.6 是 Android 的用户界面。




(a) 系统桌面




(b) 程序菜单

**图 1.6 Android 的用户界面**

# 1.2 Android 起源

## 1.2.1 开放手机联盟

说到 Android 的发展史,首先要介绍 Android 平台的推动者——开放手机联盟(Open Handset Alliance,OHA)。开放手机联盟是美国谷歌公司于 2007 年发起的一个全球性的联盟组织,目标是研发移动设备的新技术,用以大幅削减移动设备开发与推广成本。同时通过联盟各个合作方的努力,建立了移动通信领域新的合作环境,促进了创新移动设备的开发,创造了目前移动平台实现的用户体验。

开放手机联盟成立时由 34 个成员组织构成,包括电信运营商、半导体芯片商、手机硬件制造商、软件厂商和商品化公司等 5 类,涵盖移动终端产业链各个环节。目前,开放手机联盟的成员组织数目已经增加到 50 个。谷歌通过与运营商、设备制造商、开发商和其他有关各方结成深层次的合作伙伴关系,借助建立标准化、开放式的移动电话软件平台,在移动产业内形成一个开放式的生态系统。

在开放手机联盟的组织成员中,电信运营商主要有中国移动通信、KDDI(日本)、NTT DoCoMo(日本)、Sprint Nextel(美国)、T-Mobile(美国)、Telecom(意大利)、中国联通、Softbank(日本)、Telefonica(西班牙)和 Vodafone(英国),如图 1.7 所示。

开放手机联盟中的半导体芯片商有 Audience(美国)、AKM(日本)、ARM(英国)、Atheros Communications(美国)、Broadcom(美国)、Intel(美国)、Marvell(美国)、nVIDIA













SoftBank





图 1.7 电信运营商

(美国)、Qualcomm(美国)、SiRF(美国)、Synaptics(美国)、ST-Ericsson(意大利、法国和瑞典)和 Texas Instruments(美国),如图 1.8 所示。




图 1.8 半导体芯片商

开放手机联盟中的手机硬件制造商有 Acer(中国台湾)、华硕(中国台湾)、Garmin(中国台湾)、宏达电(中国台湾)、LG(韩国)、三星(韩国)、华为(中国)、摩托罗拉(美国)、索尼爱立信(日本和瑞典)和东芝(日本),如图 1.9 所示。

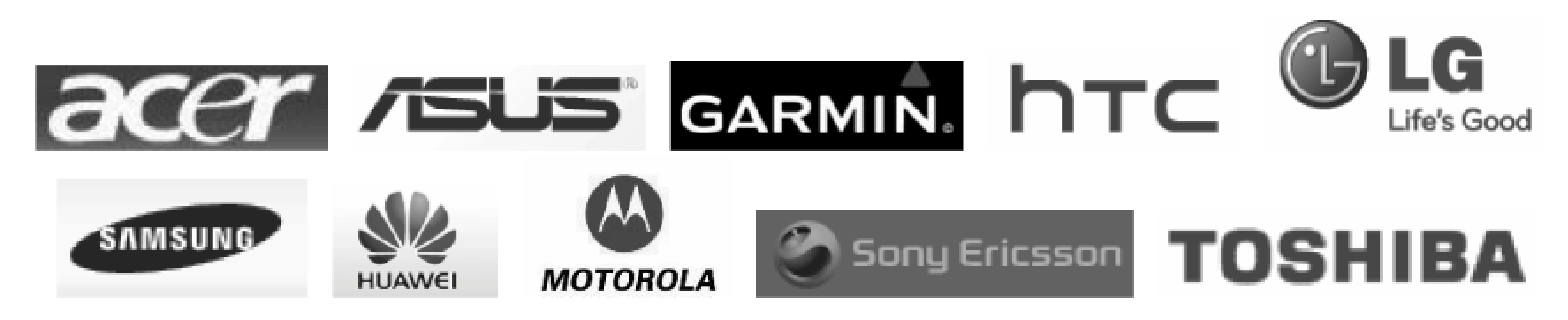



图 1.9 手机硬件制造商

开放手机联盟中的软件厂商有 Ascender Corp(美国)、eBay(美国)、谷歌(美国)、LivingImage(日本)、NuanceCommunications(美国)、Myraid(瑞士)、Omron(日本)、PacketVideo(美国)、SkyPop(美国)、Svox(瑞士)和 SONiVOX(美国),如图 1.10 所示。




图 1.10 软件厂商

开放手机联盟中的商品化公司有 Aplix Corporation(日本)、Noser Engineering(瑞士)、Borqs(中国)、TAT-The Astonishing(瑞典)、Teleca AB(瑞典)和 Wind River(美国),如图 1.11 所示。




**图 1.11 商品化公司**

## 1.2.2 Android发展史

2007年11月5日，开放手机联盟成立，由电信运营商、半导体芯片商、手机硬件制造商、软件厂商和商品化公司在内的34个组织构成，推动Android平台的研发和推广。开放手机联盟徽标如图1.12所示。



**图 1.12 开放手机联盟徽标**

2007年11月12日，发布了Android SDK预览版，这是第一个对外公布的Android SDK，为发布正式版收集用户反馈。

2008年4月17日，谷歌举办总共1000万美金的Android开发者竞赛，奖励最有创意的Android程序开发者，使Android平台在短时间积累了大量优秀的应用程序。涌现出像cab4me(出租车呼叫)、BioWallet(生物特征识别)和CompareEverywhere(实时商品查询)等极具创意的应用程序，如图1.13所示。










**图 1.13 Android开发者竞赛作品**

2008年8月28日，谷歌开通了Android Market，供Android手机下载需要使用的应用程序。程序开发人员可以将自己设计的Android软件上传到Android Market(如图1.14所示)，并决定软件是否收取费用。但在Android Market上销售软件需要向谷歌支付25美元的注册费，并在每次交易中将30%的利润支付给运营商。

2008年9月23日，发布Android SDK v1.0版，这是第一个稳定的SDK版本。SDK中分别提供了基于Windows、Mac和Linux操作系统的集成开发环境，包含完整

高效的Android模拟器和开发工具，详尽的说明文档和开发示例。程序开发人员非常容易就可以掌握Android应用程序的开发方法，同时也降低了开发手机应用程序的门槛。

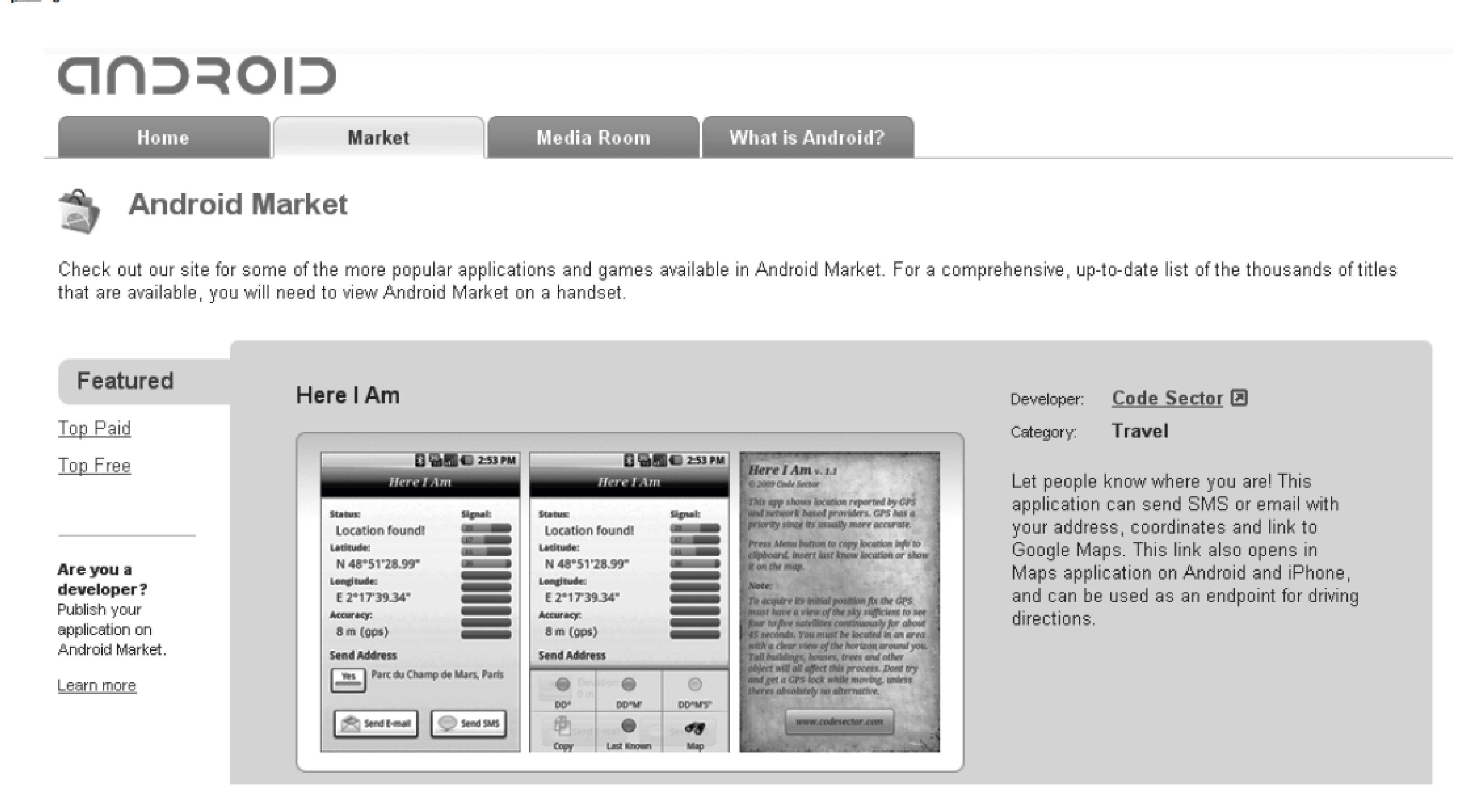



图1.14 Android Market

2008年10月22日，第一款Android手机T-Mobile G1(HTC Dream)在美国上市，由中国台湾的宏达电(HTC)制造，如图1.15所示。在硬件方面，内置528MHz的Qualcomm MSM 7201A处理器，有192MB RAM和256MB ROM的内存空间，提供侧面滑动的全键盘，支持Wi-Fi功能和内置GPS模块，支持最大8GB容量的microSD存储卡扩展容量，支持GSM/UMTS/GPRS/EDGE/HSDPA网络，在软件方面，集成了众多的应用功能，包括谷歌的地图功能、YouTube视频功能、全方位的导航定位以及360度查看浏览目标位置的功能。



图1.15 T-Mobile G1

2008年10月21日，谷歌开放了Android平台的源代码。Android作为开放源代码的手机平台，任何人或机构都可以免费使用Android，并对它做出改进。开放源代码的Android有利于创新，能够为用户提供更好的体验。同时也意味着任何厂商都可以推出基于Android的手机，且不用支付任何的许可费用。Android的源代码可以到谷歌的官方网站下载，地址是http://source.android.com/download，如图1.16所示。

2009年2月，发布Android SDK v1.1版。修正原有版本存在的缺陷，如设备休眠状态的稳定性问题、邮件冻结问题、POP3(第3代邮局协议)链接失败问题和IMAP(因特网邮件访问协议)的密码引用问题等。同时，增加了新的特性，例如当用户搜索地图或详细查看时，允许用户对地图进行评论；允许用户保存彩信的附件；为了更加便利的使用拨号盘，可以将拨号盘隐藏或显示在通话菜单中。

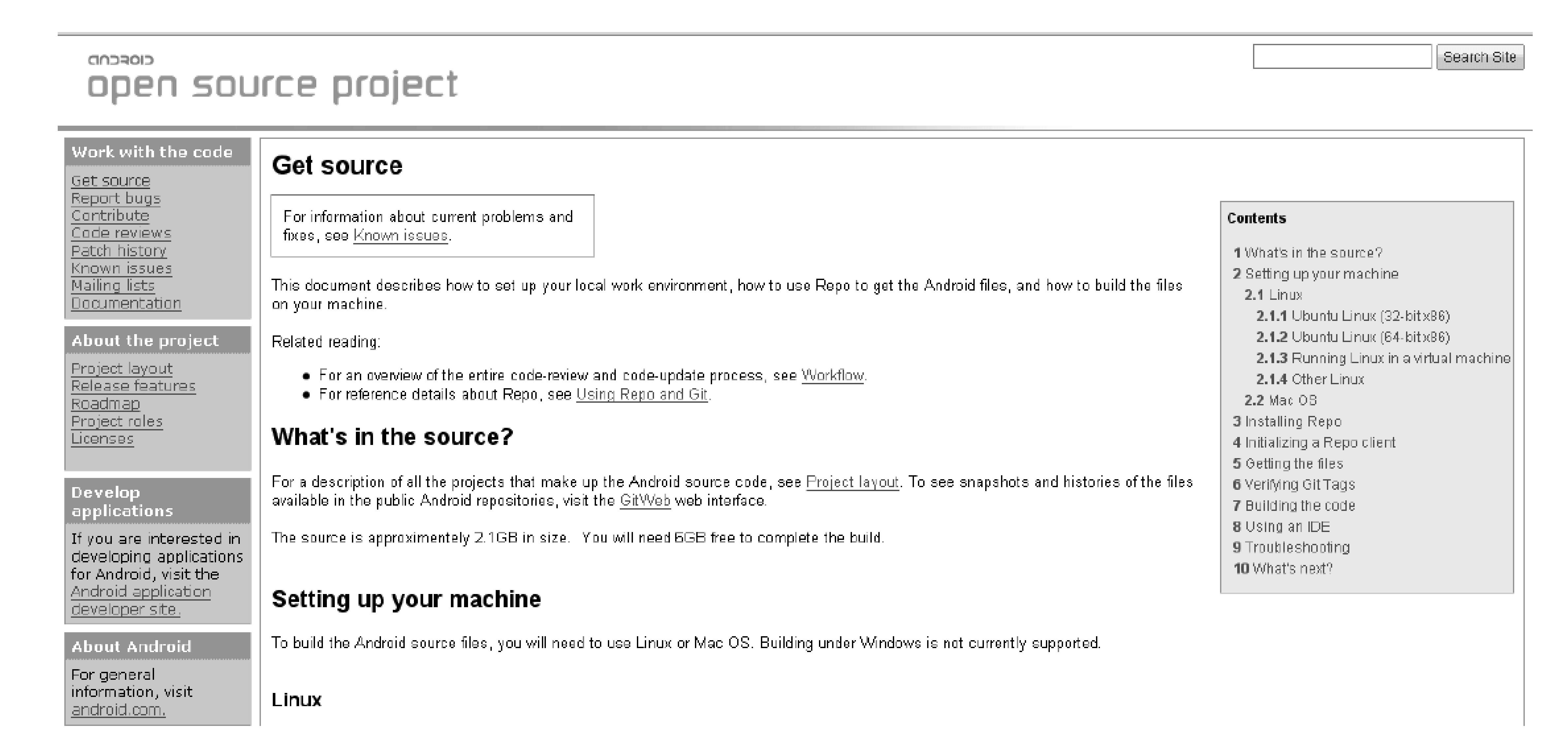



**图 1.16 Android 源代码的下载网站**

2009 年 2 月 17 日，第二款 Android 手机 T-Mobile G2(HTC Magic)正式发售，仍然由中国台湾的宏达电制造。T-Mobile G2(见图 1.17)的硬件配置与 T-Mobile G1 基本相同，不同之处主要在于将 256MB ROM 的内存空间提高到 512MB，并略微增加了的电池的容量，用以提升整机的使用时间。T-Mobile G1 放弃了影响手机尺寸的滑动全键盘设计，使 T-Mobile G2 的体积(113mm×55mm×13.65mm)比 T-Mobile G1(117.7mm×55.7mm×17.1mm)明显减小。

2009 年 4 月 15 日，发布 Android SDK v1.5 版。此版本 SDK 提升了性能表现，提高了摄像头的启动速度和拍摄速度，提高了 GPS 位置的获取速度，使浏览器的滚动更为平滑，提高了获取 Gmail 中对话列表的速度等。在新特性方面，支持了软键盘、中文显示和中文输入功能，并可以将视屏录制的内容直接上传到 Youtube。

2009 年 6 月 24 日，中国台湾的宏达电发布了第三款 Android 手机 HTC Hero(见图 1.18)。在硬件方面，使用 Qualcomm MSM 7200A 处理器，500 万像素摄像头，提供 3.5mm 的耳机插孔。在软件方面，首次支持 Adobe Flash 和支持多点触控技术，最突出的改进是使用了 HTC Sense 界面，使 HTC Hero 的界面异常美观、绚丽。




**图 1.17 T-Mobile G2**




**图 1.18 HTC Hero**

## 1.3 Android 特 征

Android 广泛支持 GSM(全球通)、DEGE 和 3G 的语音与数据业务,支持接收语言呼叫和 SMS(短信消息服务),支持数据存储共享和 IPC(进程间通信)消息机制,为地理位置服务(如 GPS)、谷歌地图服务提供易于使用的 API 函数库,提供组件复用和内置程序替换的应用程序框架,提供基于 WebKit 的浏览器,广泛支持各种流行的音频和视频文件格式,支持的格式有 MPEG4、H264、MP3、AAC、AMR、JPG、PNG 和 GIF,为 2D 和 3D 图像处理的专用的 API 库函数。

Android 系统提供了访问硬件的 API 库函数,用来简化像摄像头、GPS 等硬件的访问过程。只要支持 Android 应用程序框架的手机,对硬件访问的方法是完全一致的,因此即使将应用程序移植到不同硬件配置的手机上,也无需修改应用程序对硬件的访问方法。Android 支持的硬件包括 GPS、摄像头、网络连接、Wi-Fi、蓝牙、加速度计、触摸屏和电源管理等。

在内存和进程管理方面,Android 具有自己的运行时和虚拟机。与 Java 和 .NET 运行时不同,Android 运行时还可以管理进程的生命周期。Android 为了保证高优先级进程运行和正在与用户交互进程的响应速度,允许停止或终止正在运行的低优先级进程,以释放被占用的系统资源。Android 进程的优先级并不是固定的,而是根据进程是否在前台或是否与用户交互而不断变化的。Android 生命周期和调试的相关内容将在本书的第 4 章进行介绍。

在界面设计上,Android 提供了丰富了界面控件供使用者之间调用,从而加快了用户界面的开发速度,也保证了 Android 平台上的程序界面的一致性。Android 将界面设计与程序逻辑分离方面,使用 XML 文件对界面布局进行描述,有利于界面的修改和维护。用户界面的相关内容将在本书的第 5 章进行介绍。

Android 提供轻量级的进程间通信机制 Intent,使跨进程组件通信和发送系统级广播成为可能。通过设置组件的 Intent 过滤器,组件通过匹配和筛选机制,可以准确地获取到可以处理的 Intent。组件通信与广播消息的相关内容将在本书的第 6 章进行介绍。

Android 提供了 Service 作为无用户界面、长时间后台运行的组件。Android 是多任务系统,但受到屏幕尺寸的限制,同一时刻只允许一个应用程序是在前台运行。Service 无需用户干预,可以长时间、稳定地运行,可为应用程序提供特定的后台功能,还可以实现事件处理或数据更新等功能。后台服务相关内容将在本书的第 7 章进行介绍。

Android 支持高效、快速的数据存储方式,包括快速数据存储方式 SharedPreferences、文件存储和轻量级关系数据库 SQLite,应用程序可以使用适合的方法对数据进程保存和访问。同时,为了便于跨进程共享数据,Android 提供了通用的共享数据接口 ContentProvider,可以无需了解数据源、路径的情况下,对共享数据进行查询、添加、删除和更新等操作。数据存储与访问相关内容将在本书的第 8 章进行介绍。

## 1.4 Android体系结构

Android是基于Linux内核的软件平台和操作系统，采用了软件堆层(Software Stack)的架构，共分为4层。第1层是Linux内核，提供由操作系统内核管理的底层基础功能；第2层是中间件层，由函数库和Android运行时构成；第3层是应用程序框架层，提供了Android平台基本的管理功能和组件重用机制；第4层是应用程序层，提供了一系列核心应用程序，如图1.19所示。

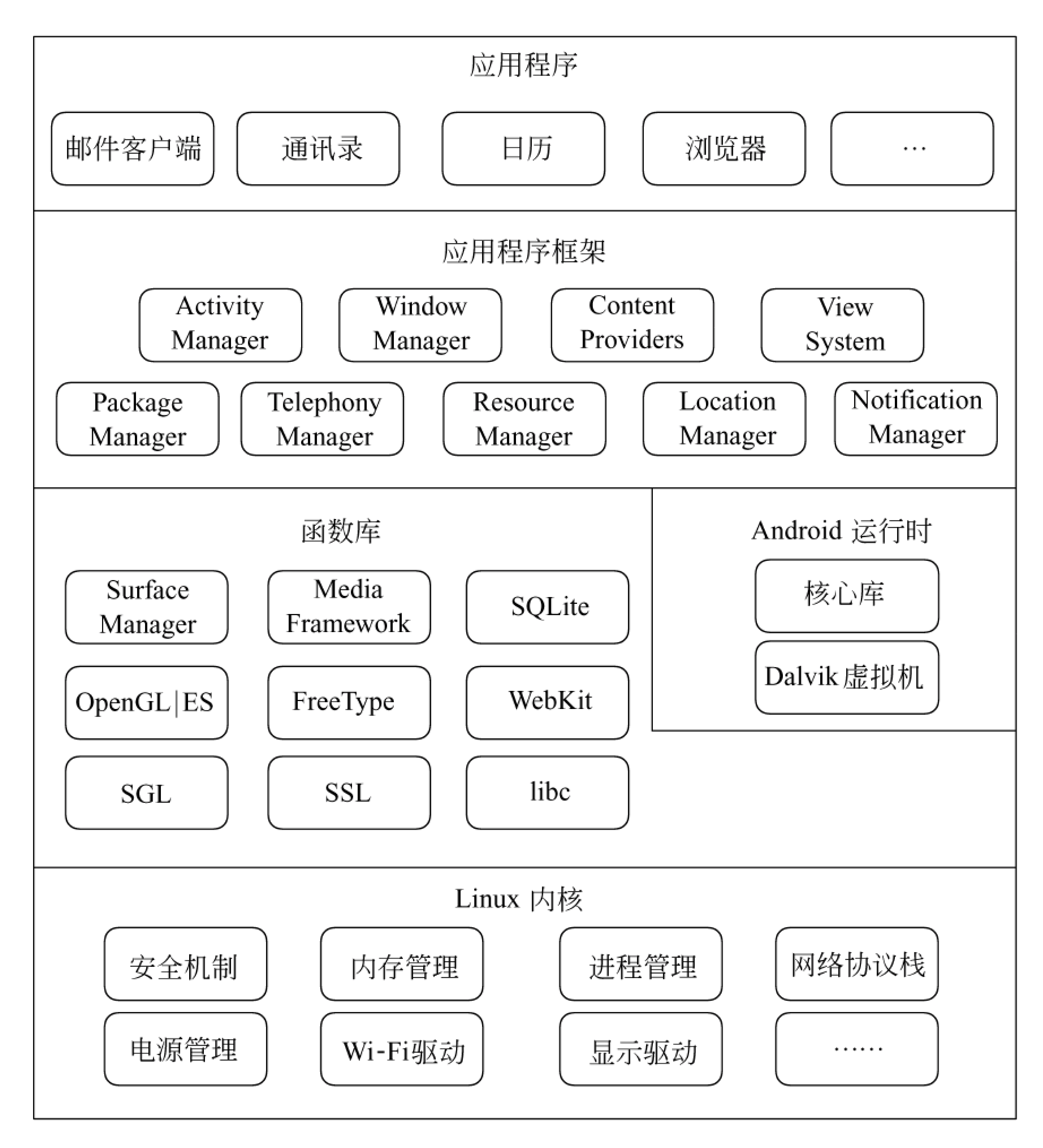



**图1.19 Android体系结构**

Android平台的底层使用的是Linux 2.6内核，硬件和其他软件堆层之间的一个抽象隔离层。提供安全机制、内存管理、进程管理、网络协议堆栈、电源管理和驱动程序等，驱动程序包括Wi-Fi驱动、声音驱动、显示驱动、摄像头驱动、闪存驱动、Binder(IPC)驱动和键盘驱动等。

函数库在Linux内涵之上，提供了一组基于C/C++的函数库，程序开发人员可以通过应用程序框架调用这些函数库。主要的函数库包括：Surface Manager，支持显示子系统的访问，为多个应用程序提供2D、3D图像层的平滑连接；Media Framework，基于

OpenCORE的多媒体框架，实现音频、视频的播放和录制功能，广泛支持多种流行的音视频格式，包括MPEG4、H.264、MP3、AAC、AMR、JPG和PNG等；SQLite，轻量级的关系数据库引擎；OpenGL ES，基于硬件的3D图像加速；FreeType，位图与矢量字体渲染；WebKit，Web浏览器引擎；SGL，2D图像引擎；SSL，数据加密与安全传输的函数库；libc，标准C运行库，它是Linux系统中底层的应用程序开发接口。

Android运行时由核心库和Dalvik虚拟机构成。核心库为程序开发人员提供了Android系统的特有函数功能和Java语言基本函数功能。Dalvik虚拟机是经过优化的多实例虚拟机，基于寄存器架构设计，实现了基于Linux内核的线程管理和底层内存管理。Dalvik虚拟机可以执行Dalvik可执行格式(.dex)，该格式适合内存和处理器速度受限的系统。

应用程序框架提供了Android平台基本的管理功能和组件重用机制，包括Activity管理、窗体管理、包管理、电话管理、资源管理、位置管理、通知消息管理、View系统和内容提供者等。ContentProvider用来共享私有数据，实现跨进程的数据访问；Resource Manager允许应用程序使用非代码资源，如图像、布局和本地化的字符串等；Notification Manager允许应用程序在状态栏中显示提示信息；Activity Manager用来管理应用程序的生命周期；Window Manager用来启动应用程序的窗体；Location Manager用来管理与地图相关的服务功能；Telephony Manager用来管理与拨打和接听电话相关的功能；Package Manager用来管理安装在Android系统内的应用程序。

应用程序提供了一系列核心应用程序，包括电子邮件客户端、浏览器、通讯录和日历等等。

## 习　题

1. 简述6种主流的手机操作系统的特点。
2. 简述Android平台的特征。
3. 描述Android平台体系结构的层次划分，并说明各个层次的作用。

# 第 2 章 chapter 2

# Android 开发环境

Android 开发环境的安装与配置是开发 Android 应用程序的第一步，也是深入 Android 平台的一个非常好的机会。通过本章的学习，读者可以完全掌握安装、配置 Android 开发环境的步骤和注意事项，熟悉 Android SDK 和 ADT 插件的用途，了解在应用程序开发过程中可能会使用到的各种工具。

**本章学习目标：**

- 掌握 Android 开发环境的安装配置方法
- 了解 Android SDK 的目录结构和示例程序
- 了解 Android 开发工具

## 21 安装 Android开发环境

Eclipse 是开发 Android 应用程序的首选集成开发环境。Eclipse 作为开源的 Java 开发环境，功能强大易于使用，而且还提供了在 Eclipse 上开发 Android 应用程序的 ADT(Android Development Toolkit)插件，简化了 Android 应用程序的开发、运行和调试。

安装 Android 开发环境，首先需要安装支持 Java 应用程序运行的 Java 开发工具包(Java Development Kit，JDK)，然后安装集成开发环境 Eclipse，最后安装 Android SDK 和 Eclipse 的 ADT 插件。

### 21.1 安装 JDK和 Eclipse

在开始下载和安装 Eclipse 之前，应该首先确认开发主机上是否已经安装了 Java 运行环境(Java Runtime Environment，JRE)。因为 Eclipse 是用 Java 语言编写的应用程序，需要 JRE 才能运行。如果 JRE 没有安装或者没有被检测到，尝试打开 Eclipse 时会有错误提示，如图 2.1 所示。

安装 JRE 的系统可以运行 Java 应用程序，但如果需要开发 Java 应用程序，则应该直接安装 JDK，因为 JDK 中包括完整的 JRE，安装 JDK 时 JRE 也被装入到操作系统中。JDK 的基本组件包括编译器，将源程序转换成字节码；打包工具，将相关的类文件打包成一个文件；文档生成器，从源码注释中提取文档；查错工具，用来进行调试和查错。




图 2.1　没有安装 JRE 的错误提示

下载 JDK 可以在浏览器输入 http://java.sun.com/javase/downloads/index.jsp，直接进入 Sun 的 JavaSE 下载页面，选择下载 JDK6 Update 14，如图 2.2 所示。

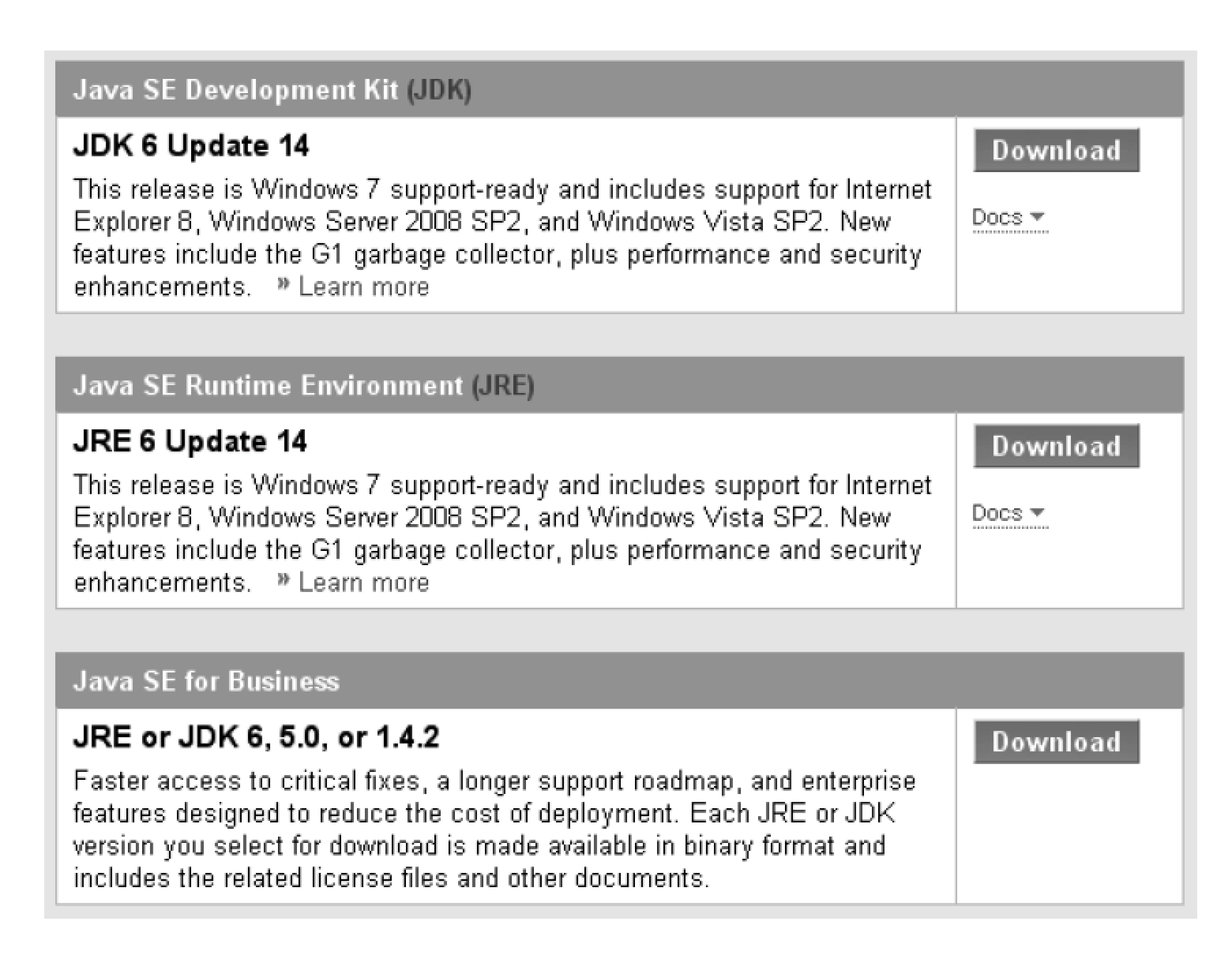



图 2.2　JavaSE 下载页面

单击 Download 按钮，进入 JKD 平台和语言选择页面，选择的平台为 Windows，语言为 Multi-language，如图 2.3 所示。

Provide Information, then Continue to Download

Select Platform and Language for your download:

Platform: Windows

Language: Multi-language

☑ I agree to the Java SE Development Kit 6u14 License Agreement

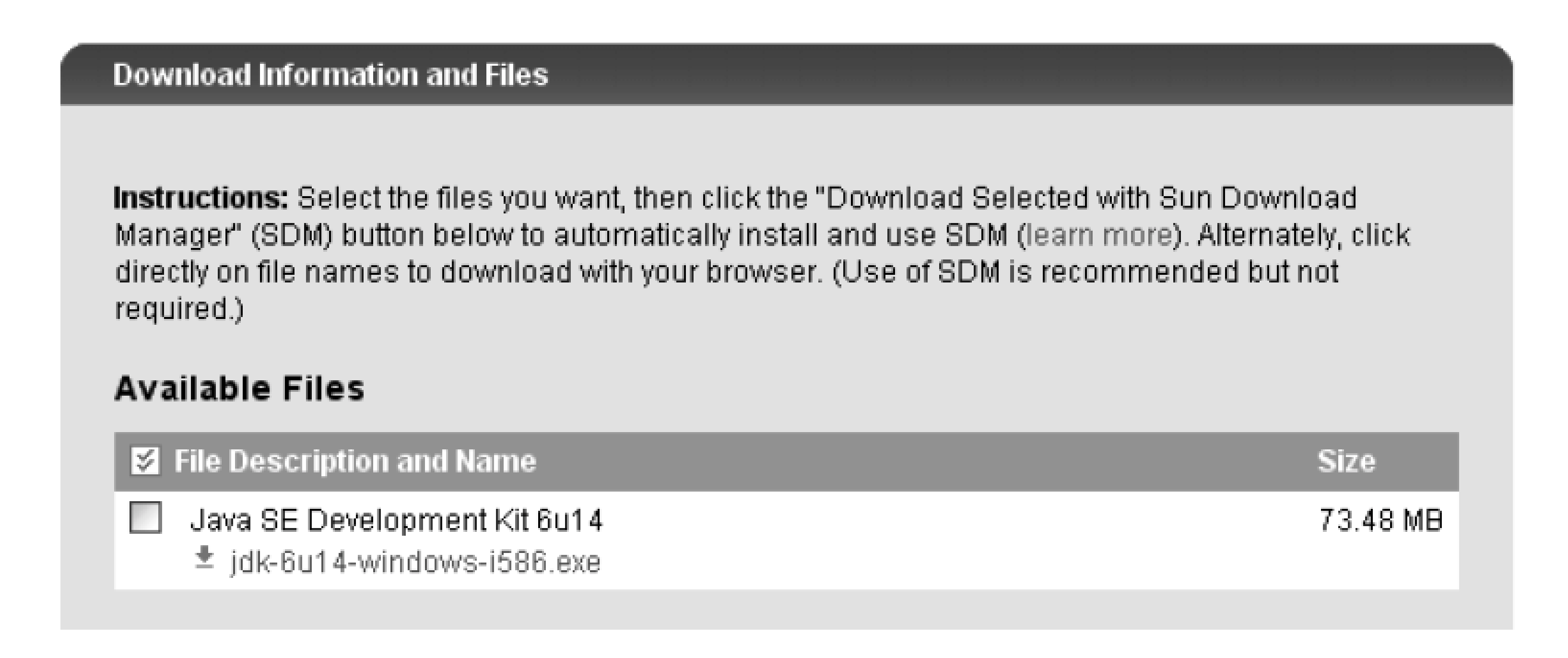
Continue »

图 2.3　JKD 平台和语言选择页面

最后进入 JDK 下载页面，下载 jdk-6u14-windows-i586.exe 文件，如图 2.4 所示。

在 JDK 的安装过程中，一般情况下保持 JDK 的默认设置即可，JDK 会安装在 C:\Program File\Java\jdk1.6.0_14\目录下，如图 2.5 所示。

JDK 安装完毕后，安装程序会提示“Java(TM) SE Development Kit 6 Update 14 已


Download Information and Files

Instructions: Select the files you want, then click the "Download Selected with Sun Download Manager" (SDM) button below to automatically install and use SDM (learn more). Alternately, click directly on file names to download with your browser. (Use of SDM is recommended but not required.)

Available Files

| File Description and Name | Size |
|---|---|
| Java SE Development Kit 6u14<br>jdk-6u14-windows-i586.exe | 73.48 MB |

图 2.4　JDK 下载页面

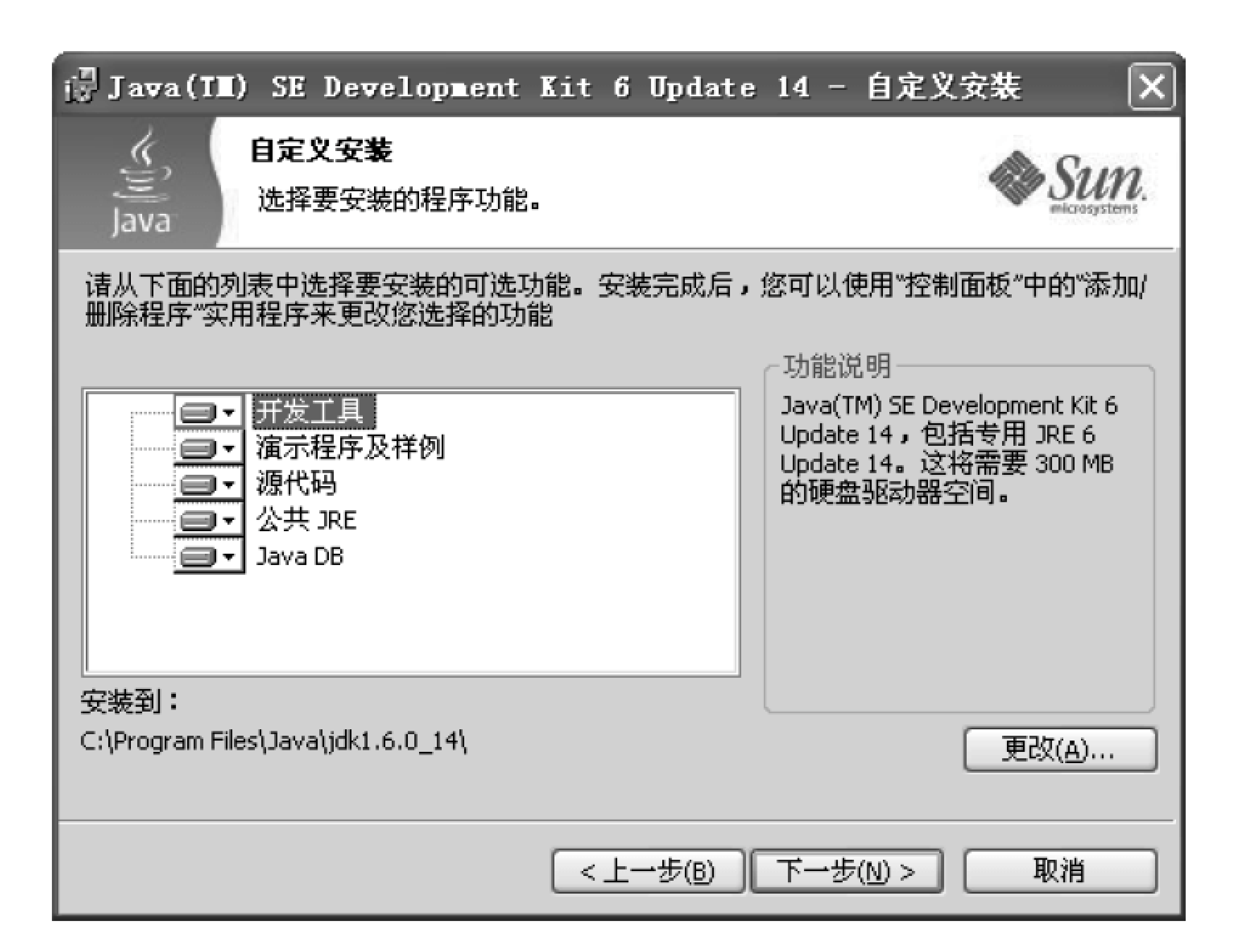



图 2.5　JDK 的默认设置

经成功安装"，如图 2.6 所示。下一步可以进行 Eclipse 的安装工作了。




图 2.6　JDK 安装成功界面

下载 Eclipse 可以在浏览器输入 http://www.eclipse.org/downloads,进入 Eclipse 网站的主页面,选择下载"Eclipse IDE for Java Developers (92MB)"的 Windows 版本,如图 2.7 所示。




图 2.7 Eclipse 下载页面

下载的 Eclipse 是一个 ZIP 文件,文件名为 eclipse-java-galileo-win32.zip,直接将 ZIP 文件中的 eclipse 文件夹解压缩到指定的目录中即可。双击目录中的 eclipse.exe 文件,出现 Eclipse 集成开发环境的启动画面,如图 2.8 所示。如果没有出现启动画面,可以尝试重新启动计算机。

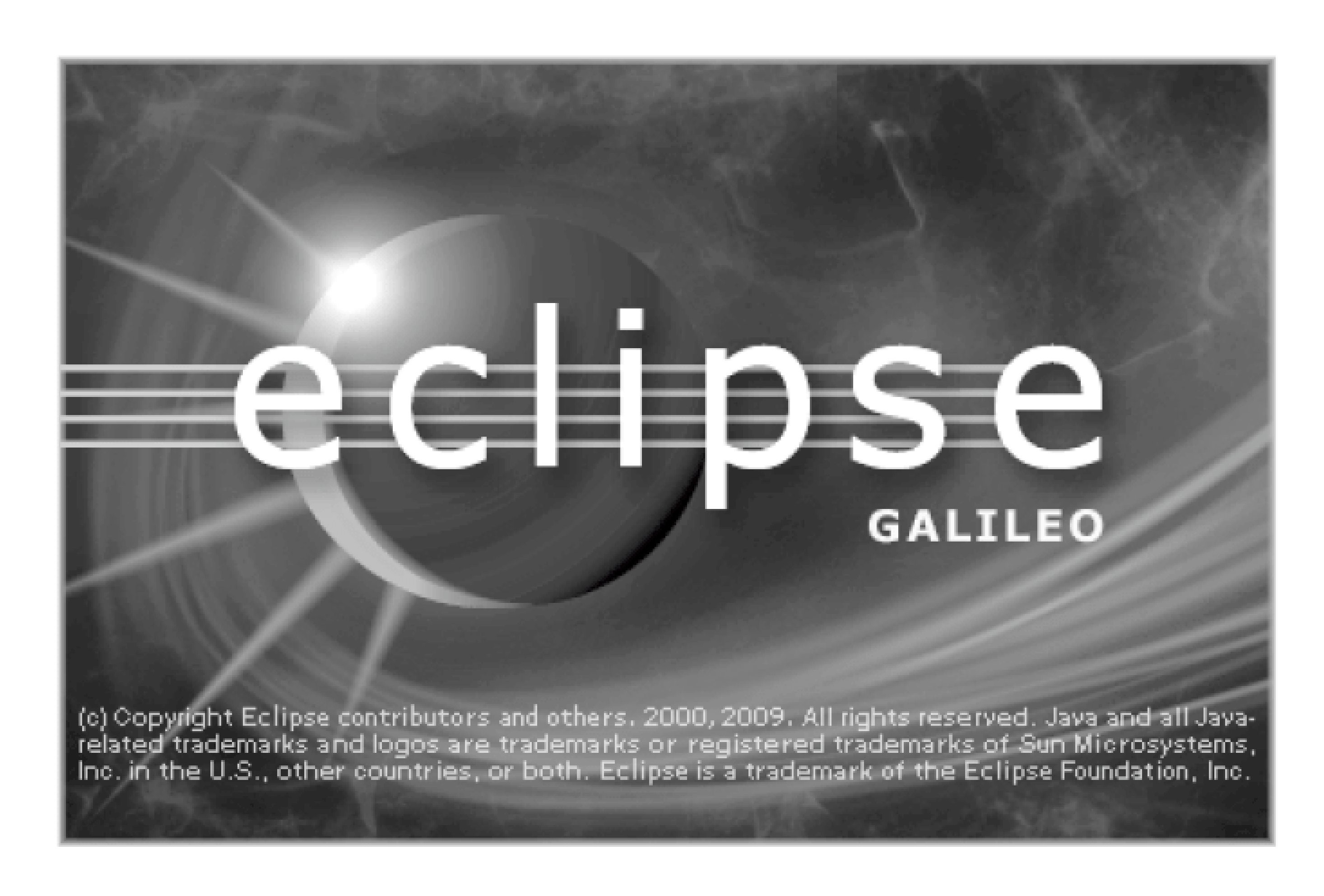



图 2.8 Eclipse 启动画面

Eclipse 启动时会提示选择一个默认工作目录,以后创建的工程将保存在这个工作目录下。工作目录的默认路径是用户目录,建议用户选择其他位置保存 Android 工程。作者选择的工作目录是 E\Android\workplace,如图 2.9 所示。建议选择复选框 Use this as the default and do not ask again,将选择的工作目录作为默认工作目录,不必每次启动时都重新选择工作目录。

正常启动后的 Eclipse 集成开发环境如图 2.10 所示。

到此为止,Eclipse 和 JDK 已经安装完毕,但在创建 Android 工程之前,还需要下载和安装 Android SDK 和 ADT 的插件,并完成 Eclipse 的相关设置。

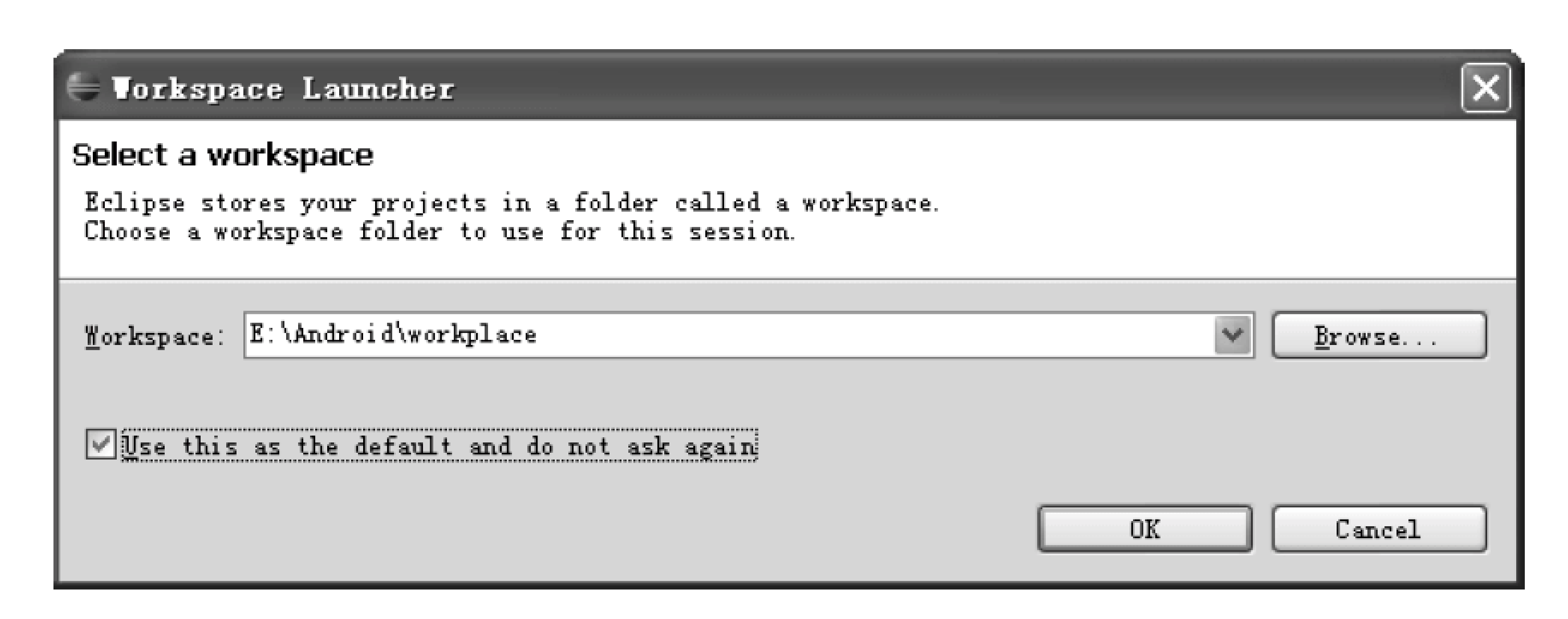



图 2.9 Eclipse 工作目录设定

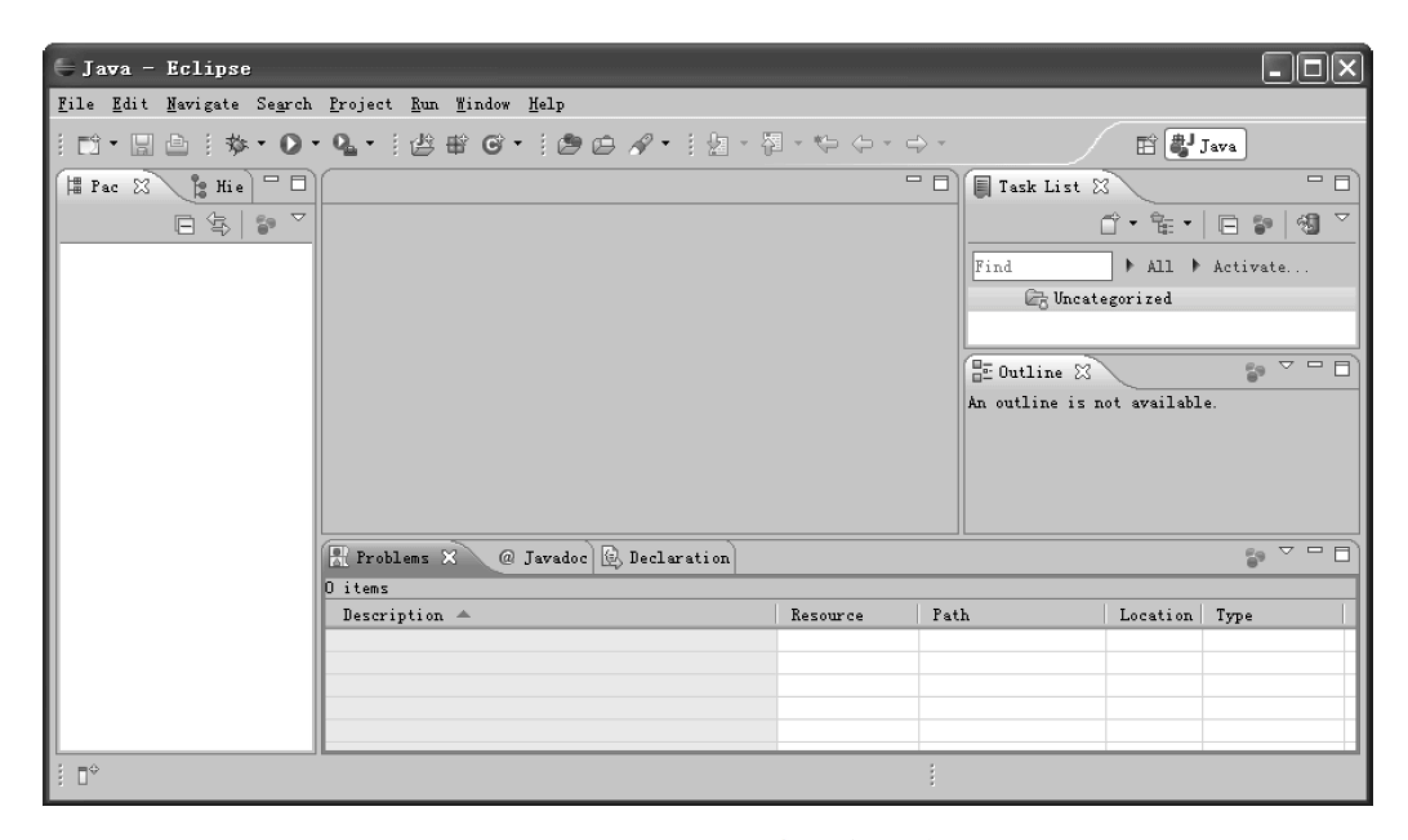



图 2.10 Eclipse 集成开发环境

## 2.1.2 安装 Android SDK

Android SDK 是 Android 软件开发工具包(Android Software Development Kit),是 Google 公司为了提高 Android 应用程序开发效率、减少开发周期而提供的辅助开发工具、开发文档和程序范例。

Android SDK 可以从 Google 的中文 Android 开发网站上下载,网站的地址是 http://code.google.com/android。开发网站不仅能够下载最新的 Android SDK,还有许多有价值的内容,如开发文档、常见问题解答等,图 2.11 是中文 Android 开发网站的主页面。

在主页面上点击"下载 SDK"超链接,则可以进入最新版本的 Android SDK 下载页面,如图 2.12 所示。根据操作系统不同,选择下载不同版本的 Android SDK。选择下载 Windows 版本的 Android SDK,文件名为 android-sdk-windows-1.5_r3.zip。

Android SDK 下载完成后,将 ZIP 压缩包中的目录解压到硬盘中。本文将 Android SDK 解压到 E:\Android\android-sdk-windows-1.5_r3 目录中。Android SDK 不需要安

装，但在 Eclipse 中开发 Android 程序前，要把 Android SDK 与 Eclipse 开发环境关联起来，具体的关联方法在 2.1.3 节进行介绍。




**图 2.11 中文 Android 开发网站的主页面**

**Download Android 1.5 SDK, Release 3**

*July 2009*

Before downloading, please read the System Requirements document. As you start the download, you will also need to review and agree to the Terms and Conditions that govern the use of the Android SDK.

| Platform | Package | Size | MD5 Checksum |
|---|---|---|---|
| Windows | android-sdk-windows-1.5_r3.zip | 191477853 bytes | 1725fd6963ce69102ba7192568dfc711 |
| Mac OS X (intel) | android-sdk-mac_x86-1.5_r3.zip | 183024673 bytes | b1bafdaefdcec89a14b604b504e7daec |
| Linux (i386) | android-sdk-linux_x86-1.5_r3.zip | 178117561 bytes | 350d0211678ced38da926b8c9ffa4fac |

For more information on this SDK release, read the Release Notes.

**图 2.12 Android SDK 下载页面**

## 2.1.3 安装 ADT 插件

ADT 插件是 Eclipse 集成开发环境的定制插件，为开发 Android 应用程序提供了一个强大的、完整的开发环境，可以快速建立 Android 工程，用户界面和基于 Android API 的组件，还可以使用 Android SDK 提供的工具进行程序调试，对 apk 文件进行签名等等。

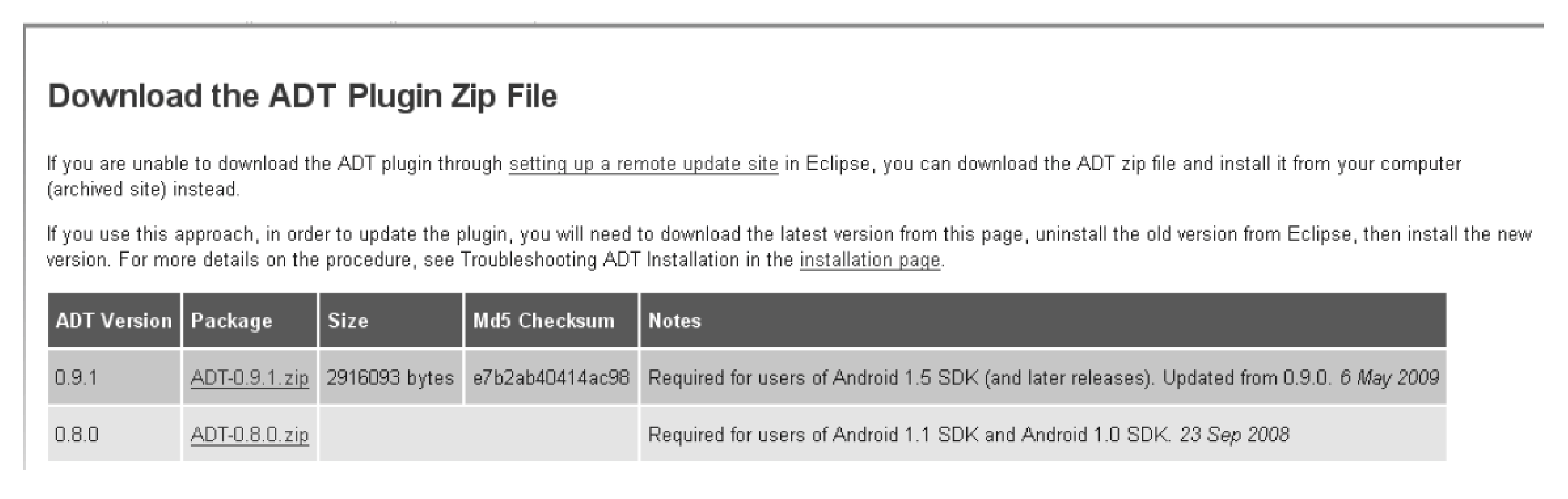
一般情况下，推荐使用安装 ADT 插件的 Eclipse 开发 Android 应用程序，因为这是最为便捷、快速的开发环境。在 Eclipse 和 Android SDK 正确安装后，就可以下载并安装 ADT 插件了。

安装 ADT 插件有两种方法，一种方法是手动下载 ADT 插件的压缩包，然后在 Eclipse 中进行安装；另一种方法是在 Eclipse 中输入插件的下载地址，由 Eclipse 自动完成下载和安装工作。推荐使用第一种方法，下载到本地的 ADT 插件压缩包可以再次使用，且安装成功的几率很高。下面对两种安装 ADT 插件的方法进行介绍。

1）方法一——手动下载 ADT 插件

手动下载 ADT 插件压缩包的网址是 http://dl-ssl.google.com/android/eclipse/。如果上面提供的网页无法打开，可以在 Android SDK 帮助文档中找到下载页面，下载页

面的位置是<Android SDK>/docs/sdk/adt_download.html,网页打开后的 ADT 插件的下载页面如图 2.13 所示。



Download the ADT Plugin Zip File

If you are unable to download the ADT plugin through setting up a remote update site in Eclipse, you can download the ADT zip file and install it from your computer (archived site) instead.

If you use this approach, in order to update the plugin, you will need to download the latest version from this page, uninstall the old version from Eclipse, then install the new version. For more details on the procedure, see Troubleshooting ADT Installation in the installation page.

| ADT Version | Package | Size | Md5 Checksum | Notes |
|---|---|---|---|---|
| 0.9.1 | ADT-0.9.1.zip | 2916093 bytes | e7b2ab40414ac98 | Required for users of Android 1.5 SDK (and later releases). Updated from 0.9.0. *6 May 2009* |
| 0.8.0 | ADT-0.8.0.zip | | | Required for users of Android 1.1 SDK and Android 1.0 SDK. *23 Sep 2008* |

**图 2.13　ADT 插件的下载页面**

在下载页面中选择下载 ADT-0.9.1.zip 文件,下载过程仍需要访问互联网,实际的下载地址为 http://dl-ssl.google.com/android/ADT-0.9.1.zip。

在下载 ADT 插件压缩包后,启动 Eclipse,选择 Help→Install New Software,打开 Eclipse 的插件安装界面,如图 2.14 所示。

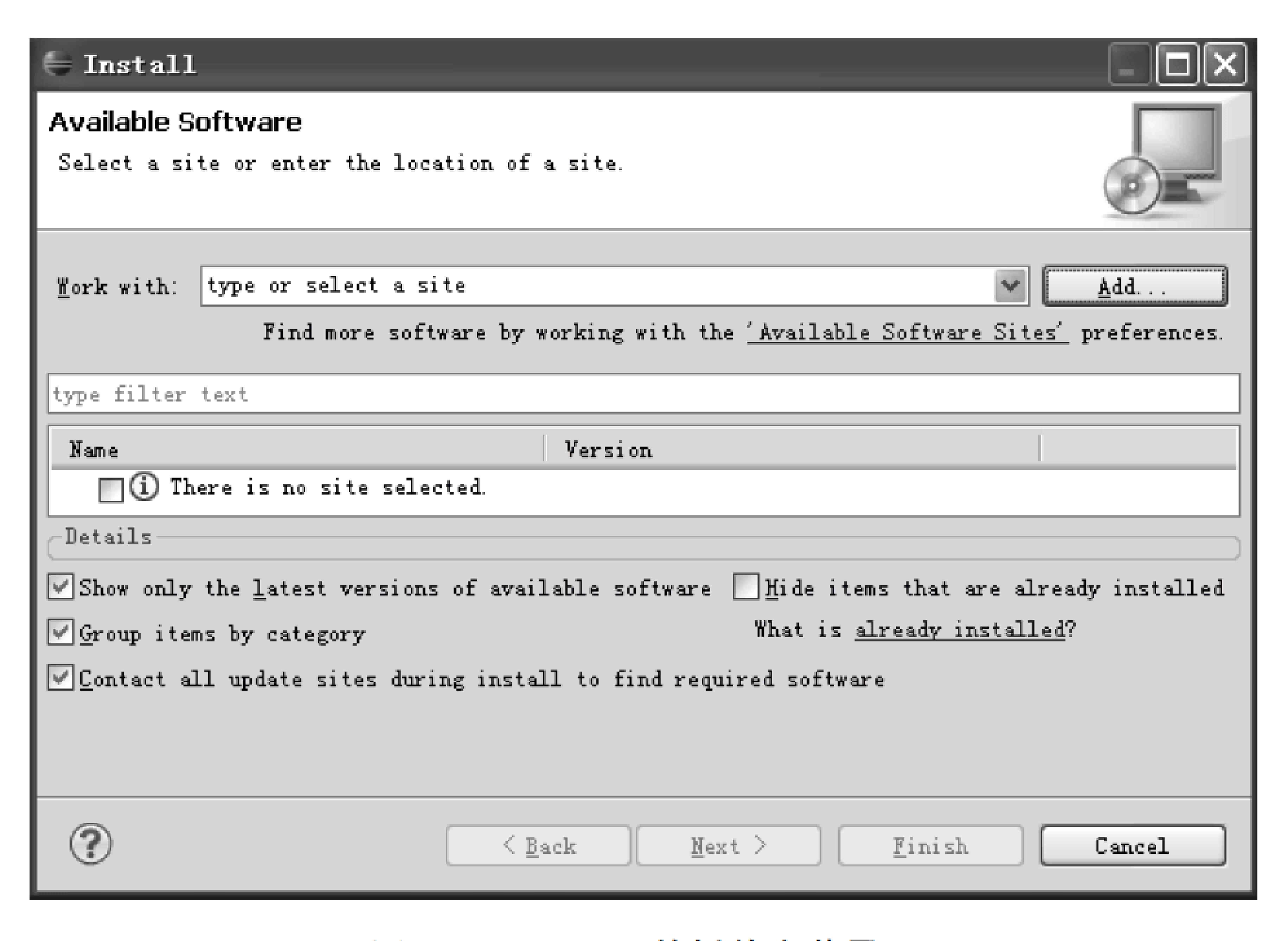



**图 2.14　Eclipse 的插件安装界面**

单击 Add 按钮,弹出 Add Site 界面,如图 2.15 所示。然后再次单击 Archive 按钮选择 ADT 插件压缩包在本地磁盘中的保存位置。正确选择保存位置后,ADT 插件压缩包的保存位置将显示在 Location 文本框中。

正确选择 ADT 插件压缩包的保存位置后,Eclipse 的插件安装界面会出现 ADT 插件安装选项,如图 2.16 所示。选中 Android DDMS 复选框和 Android Development Tools 复选框,然后单击 Next 按钮进入 ADT 插件许可界面。

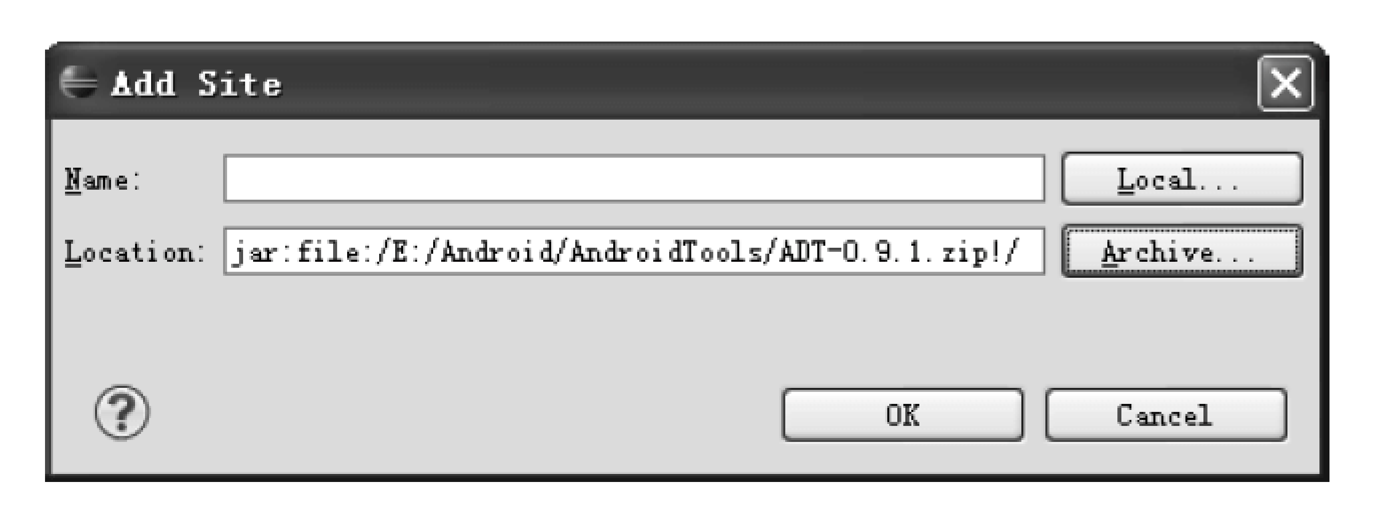



图 2.15　Add Site 界面

| Name | Version |
| --- | --- |
| Developer Tools | |
| Android DDMS | 0.9.1.v200905011822-1621 |
| Android Development Tools | 0.9.1.v200905011822-1621 |

图 2.16　ADT 插件安装选项

在 ADT 插件安装前，还需要认可 ADT 插件所使用的许可证。ADT 插件使用许可界面如图 2.17 所示，选择 I accept the terms of the license agreements 单选项即可。

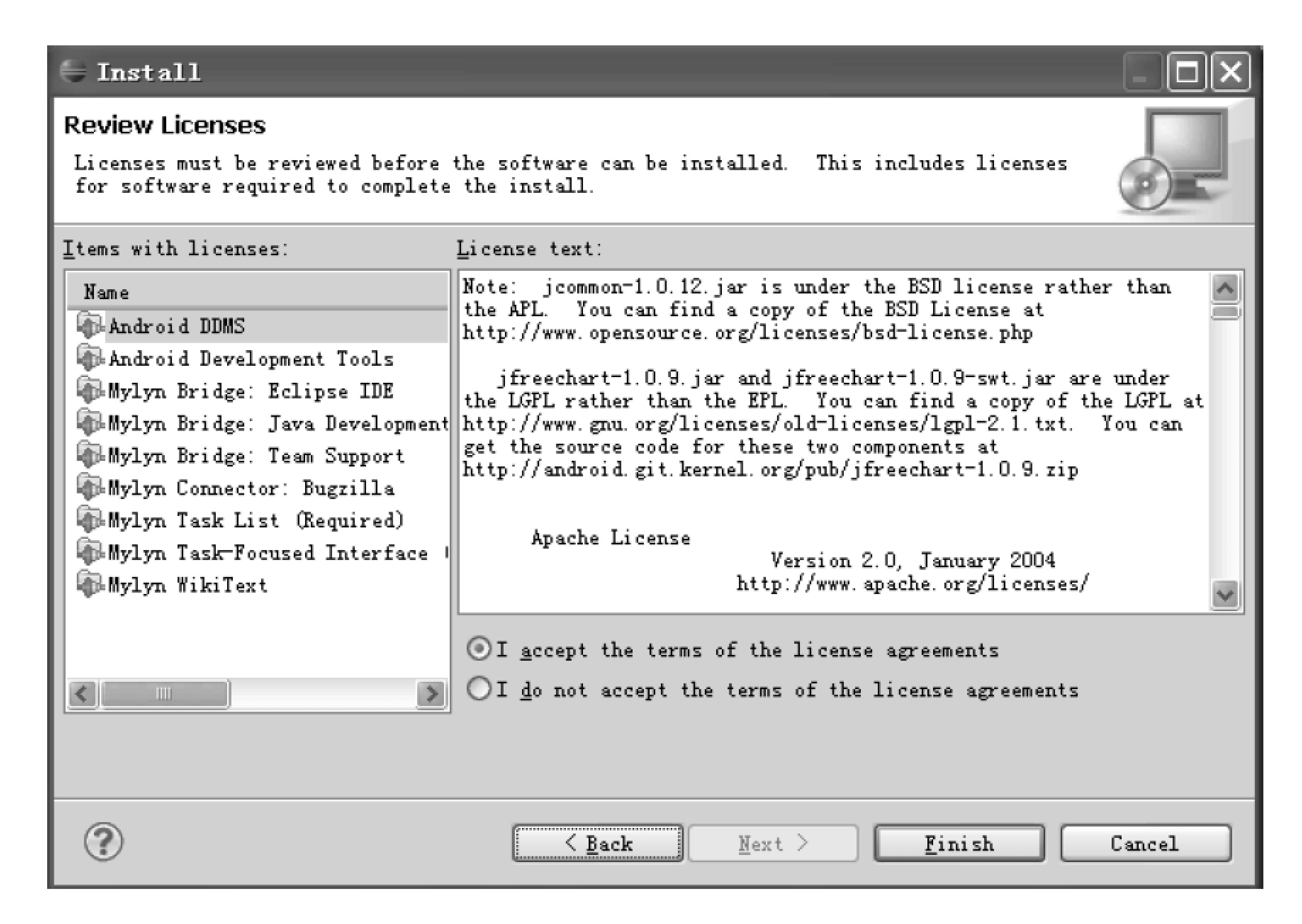



图 2.17　ADT 插件使用许可界面

整个安装过程会持续几分钟，安装结束后会出现 Eclipse 重启提示界面，如图 2.18 所示。建议选择 Yes 按钮重新启动 Eclipse，使 ADT 插件生效。




图 2.18　Eclipse 重启提示界面

2）方法二——自动下载 ADT 插件

Eclipse 自动下载 ADT 插件的方法与手动方法安装相似，不同之处在于不需要手工

到网站上下载 ADT 插件压缩包，而直接在 Add Site 界面中输入压缩包的下载地址即可，如图 2.19 所示。

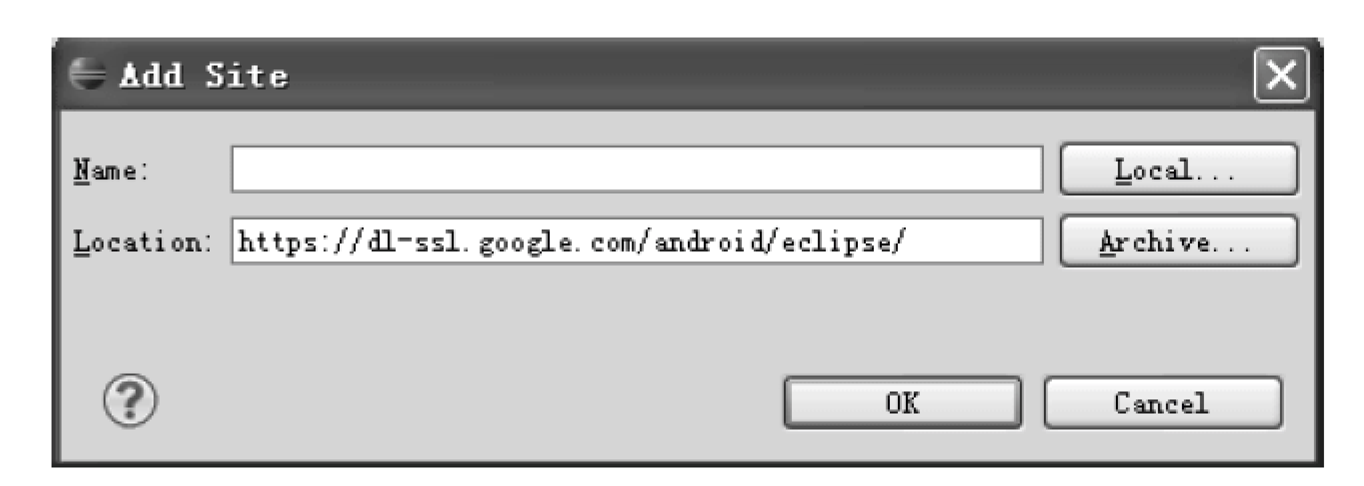



**图 2.19　Add Site 界面**

在 Location 中输入 https://dl-ssl.google.com/android/eclipse/，单击 OK 按钮后，Eclipse 会自动搜索可安装的插件。在网络连通的状态下，Eclipse 会显示如图 2.16 的 ADT 插件安装选项，之后的安装过程与“手动下载 ADT 插件”的方法完全一致。

在 ADT 插件安装完毕后，进入配置 Android 开发环境的最后一步：设置 Android SDK 的保存路径。首先选择 Windows→Preferences 命令打开 Eclipse 的配置界面，然后在 SDK Location 中输入 Android SDK 的保存路径，最后单击 Apply 按钮使配置生效，如图 2.20 所示。

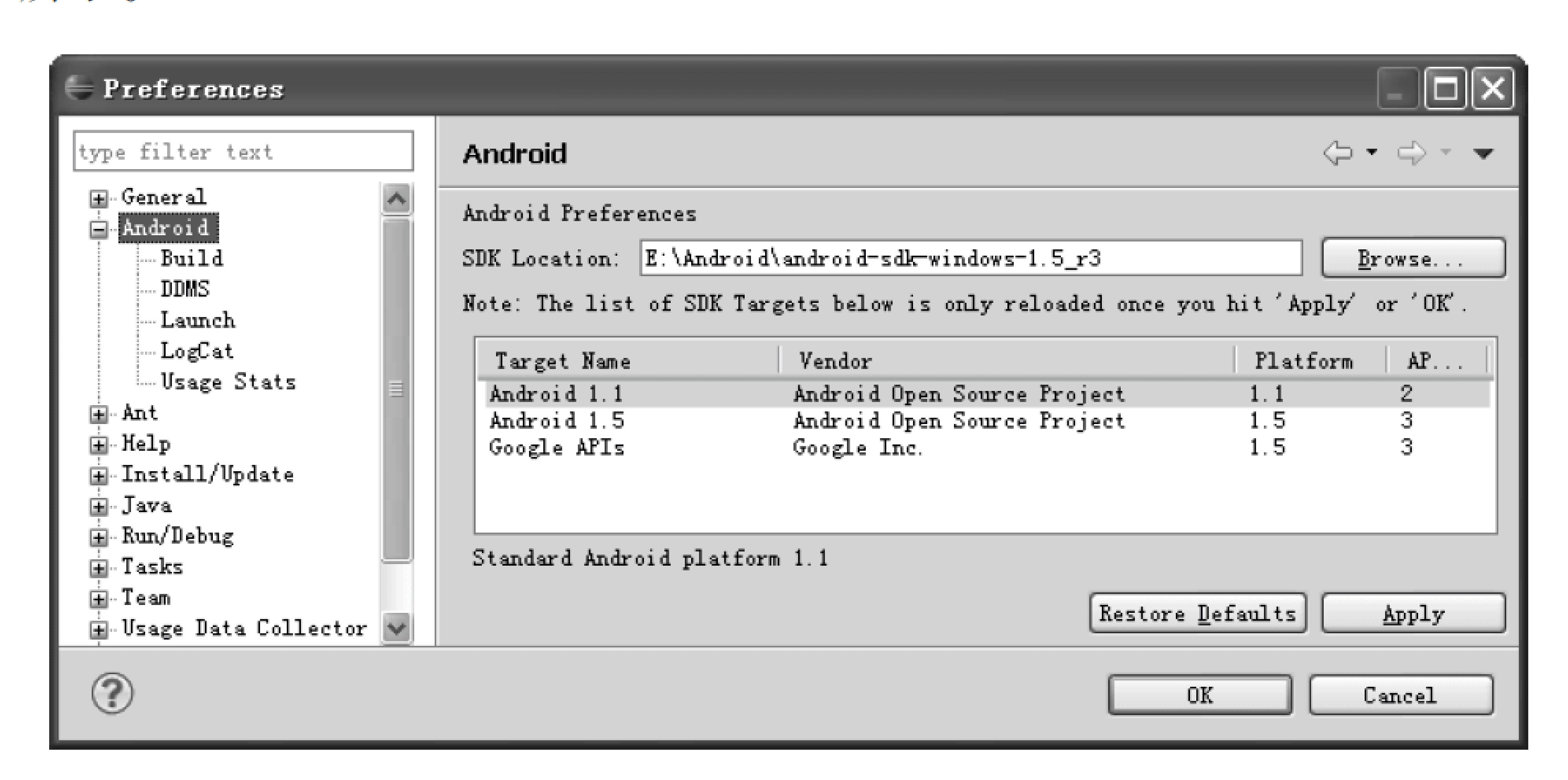



**图 2.20　Eclipse 配置界面**

至此，Android 应用程序的开发环境就安装完成了，后续的内容将对 Android SDK 的目录结构、示例程序和开发工具进行介绍。

# 22　Android SDK

Android SDK 是程序开发人员学习和开发 Android 程序的宝贵资源，不仅提供了开发所必备的调试、打包和仿真工具，还提供了详尽的说明文档和简单易懂的开发示例。

## 221　目录结构

在 Android SDK 安装到本地磁盘后，可以在文件系统中查看到 Android SDK 的目录结构。Android SDK 的目录结构如下所示：

```
(+)add-ons
   (+)google_apis-3
   (-)README.txt
(+)docs
(+)platforms
   (+)android-1.1
   (+)android-1.5
(+)tools
(+)usb_driver
   (+)amd64
   (+)x86
(-)documentation.html
(-)RELEASE_NOTES.html
```

其中,add-one 目录下的是 Google 提供地图开发的库函数,支持基于 Google Map 的地图开发。docs 目录下的是 Android SDK 的帮助文档,通过根目录下的 documentation.html 文件启动,Android SDK 帮助文档的首页如图 2.21 所示。platforms 目录中存在两个子目录 android-1.1 和 android-1.5,分别用来保存 1.1 版本和 1.5 版本的 Android SDK 的库函数、外观样式、程序示例和辅助工具等内容。tools 目录下的是通用的 Android 开发和调试工具。usb_driver 目录下保存了用于 amd64 和 x86 平台的 USB(通用串行总线)驱动程序。RELEASE_NOTES.html 是 Android SDK 的发布说明。




Welcome to the Android SDK!

If you've just downloaded the SDK, then continue with Installing the Android SDK.

If you're upgrading from a previously installed version, then refer to the Upgrading guide.

Once you've completed the SDK installation, you can start learning about development on the Android framework by reading the Developer's Guide. The SDK package also includes a wide variety of code samples specific to each Android platform. You can find the samples at this location:






Please note that you are currently viewing a local, offline version of the Android developer documentation. The offline documentation offers the same content and features as the online documentation, but since the search capability is network-based, it will not work while you are offline. Autocompletion for API packages and classes, also offered from the search box, is available both online and offline and loads the reference documentation for the item you select.

For the most current, fully searchable documentation, please visit

http://developer.android.com.

Except as noted, this content is licensed under Creative Commons Attribution 2.5. For details and restrictions, see the Content License.
Site Terms of Service - Privacy Policy - Brand Guidelines

**图 2.21 Android SDK 帮助文档**

Android SDK 帮助文档内容非常丰富,详细介绍了 Android 系统中所有 API 函数的使用方法,尤其帮助文档中的开发指南(Dev Guide),系统地介绍了 Android 应用程序的开发基础、用户界面、资源使用、数据存储、音视频功能、集成开发环境和开发工具等内

容，对于学习 Android 程序开发具有指导意义。

## 222 示例程序

在<Android SDK>\platforms\android-1.5\samples 目录下，存放了针对 Android SDK 1.5 版本的多个示例程序。这些示例程序多数并不复杂，但可以从不同方面展示 Android SDK 所提供的丰富功能。

1）HelloActivity 示例

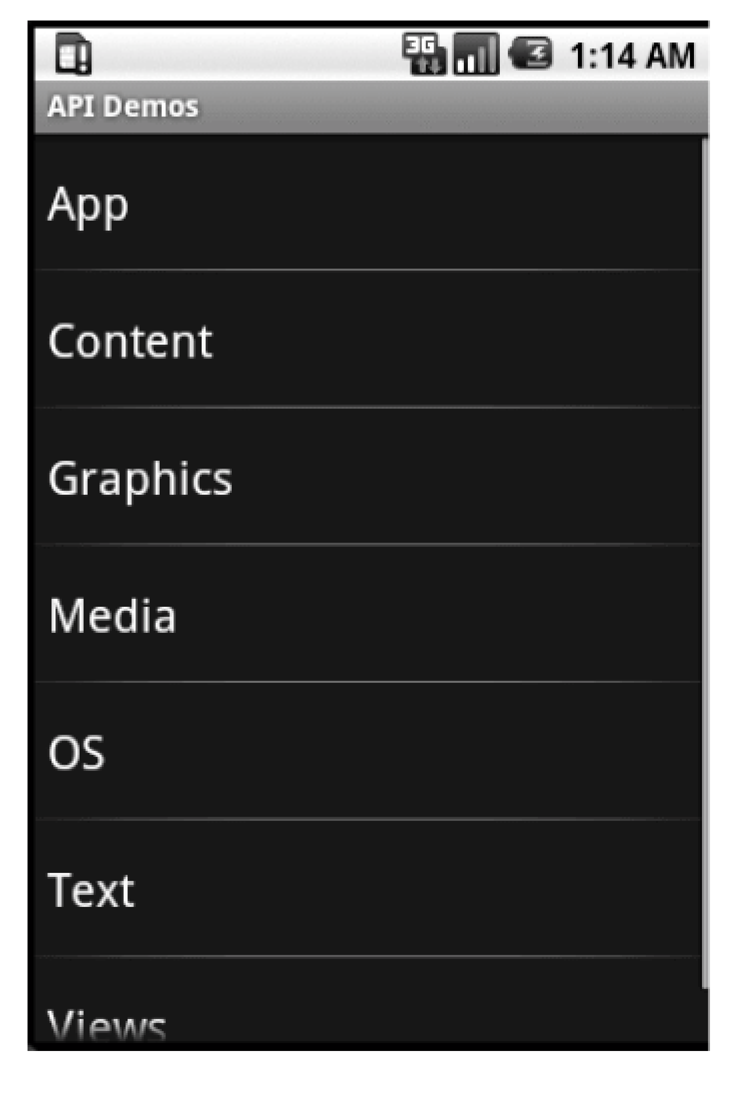
HelloActivity 是最简单的 Android 程序示例。虽然程序简单，却可以完整展示 Android 工程的文件结构和调试过程。程序开发人员可以参考 HelloActivity 示例，完成第一个 Android 应用程序。HelloActivity 示例如图 2.22 所示。

2）ApiDemos 示例

ApiDemos 示例提供了 Android 平台上多数 API 的使用方法，涉及系统、资源、图形、搜索、语音识别和用户界面等方面。程序开发人员可以在 Android 应用程序开发过程中参考 ApiDemos 示例，但该示例的代码文件众多，结构略显混乱，给参考和学习带来一定的不便。ApiDemos 示例如图 2.23 所示。




图 2.22 HelloActivity 示例



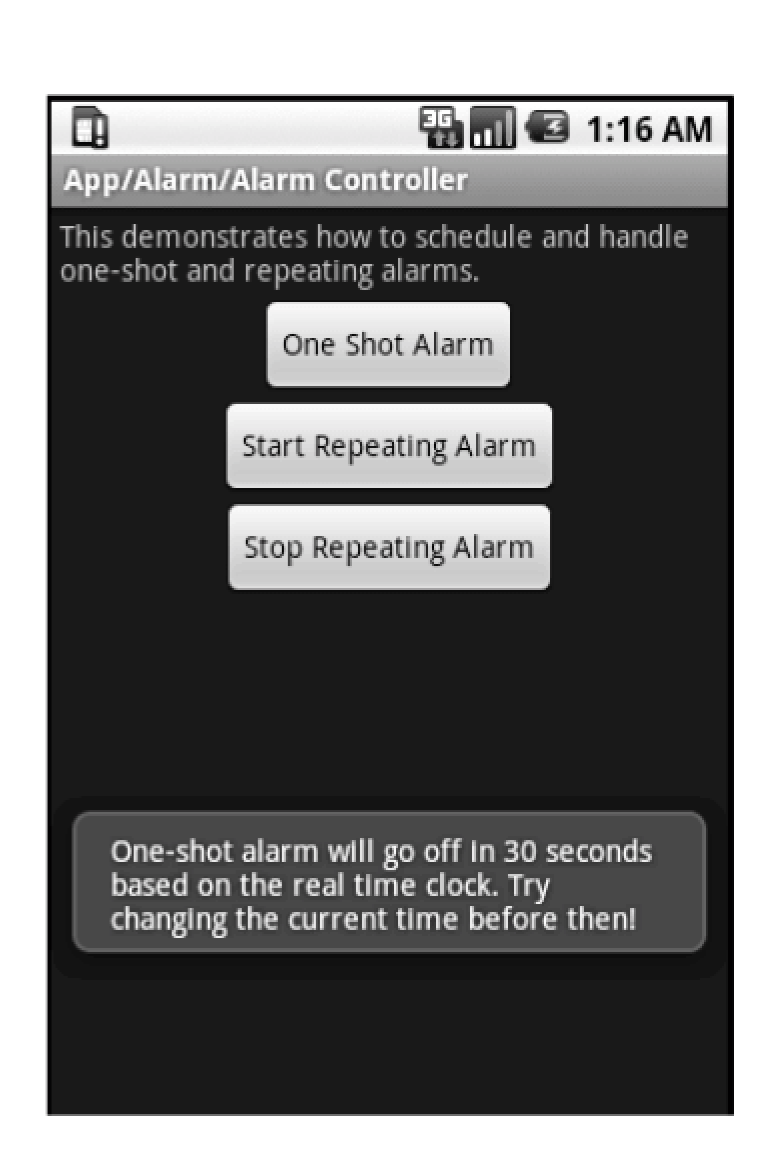




图 2.23 ApiDemos 示例

3）SkeletonApp 示例

SkeletonApp 示例是一个界面演示程序，说明了如何使用布局和界面控件设计用户界面，以及如何在界面中添加菜单和处理菜单事件。SkeletonApp 示例如图 2.24 所示。

4）NotesList 示例

NotesList 示例是一个记事本程序，可以将文字内容保存在记事本程序中，并支持添加和删除记事本操作。NotesList 示例说明了如何进行复杂程序设计，以及如何使用 SQLite 数据库保存数据和 ContentProvider 共享数据。NotesList 示例如图 2.25 所示。




图 2.24 SkeletonApp 示例

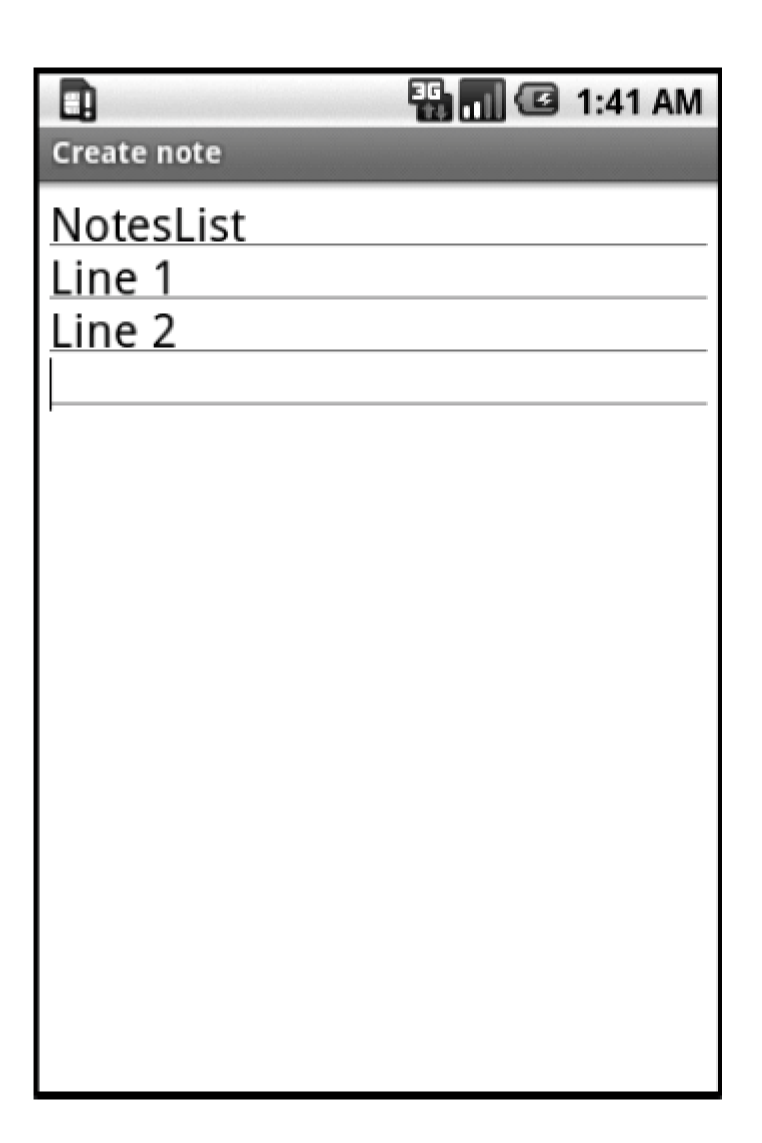
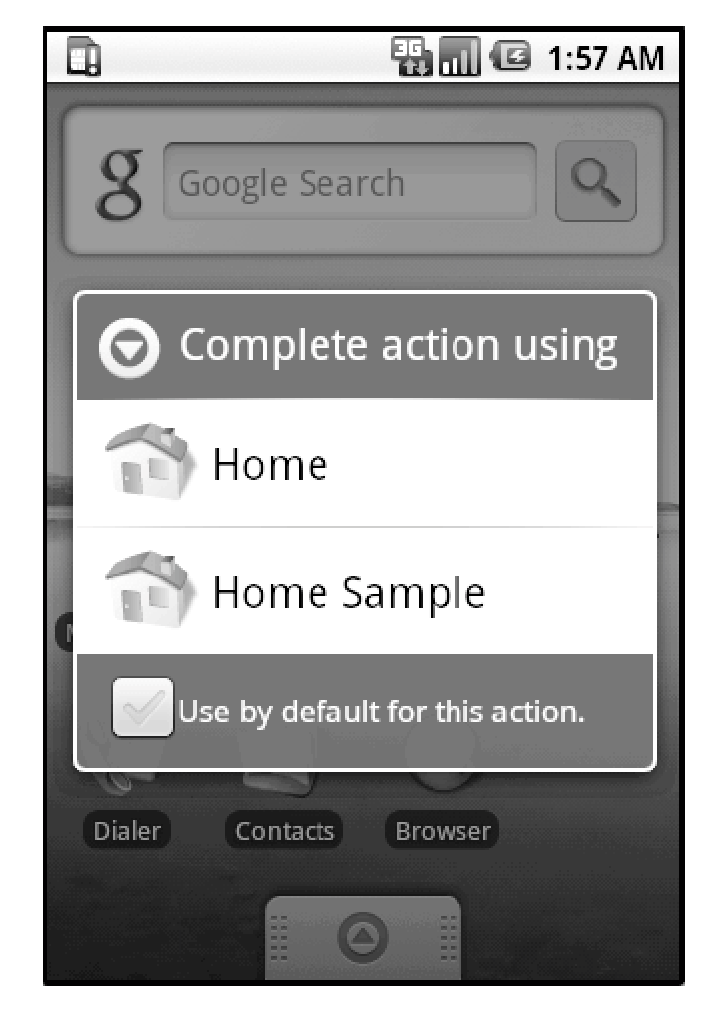



图 2.25 NotesList 示例

5）Home 示例

Home 示例是一个桌面主题程序，可以将自定义的桌面主题注册到系统中，用户可以通过单击 HOME 按钮选择不同的桌面主题。此示例说明了如何进行桌面主题程序的开发，以及在开发过程中需要注意的事项。Home 示例如图 2.26 所示。



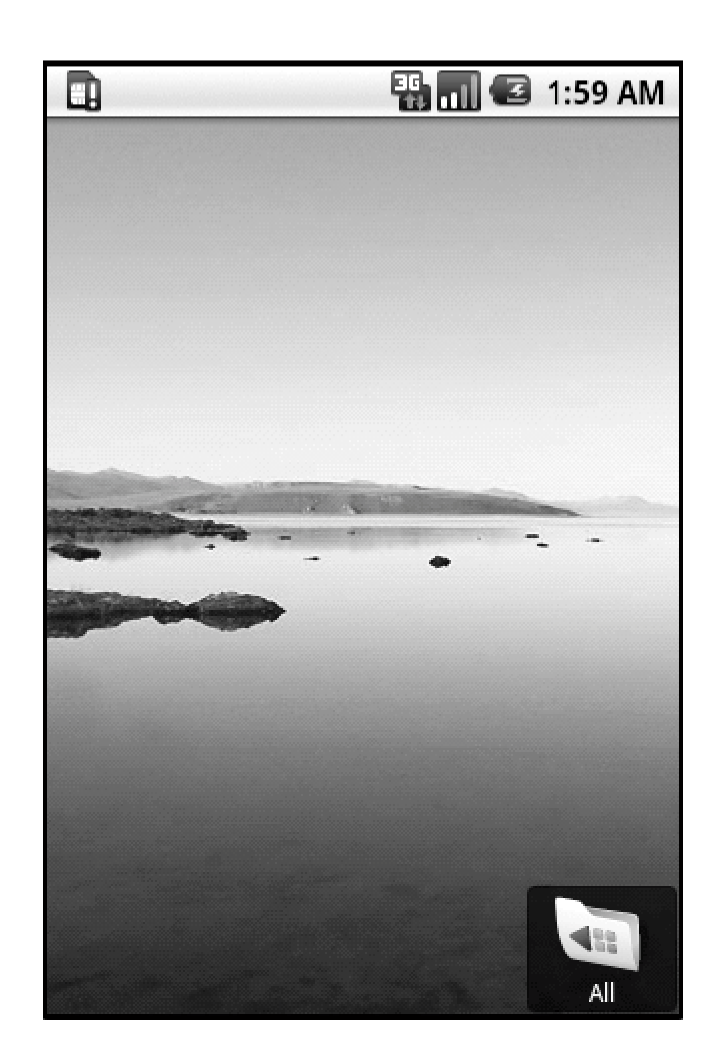




图 2.26 Home 示例

6）SoftKeyboard 示例

SoftKeyboard 示例是一个软键盘程序，在用户将焦点切换到输入框后，软键盘会自动弹出，用户可以使用软键盘进行英文文字输入。SoftKeyboard 示例如图 2.27 所示。

7）Snake 示例

Snake 示例是贪吃蛇程序，一个经典的小游戏，可以通过导航键控制贪吃蛇的前进方向。该示例演示了如何在 Android 系统中进行游戏开发，对进行游戏开发的程序人员具有一定的参考价值。Snake 示例如图 2.28 所示。

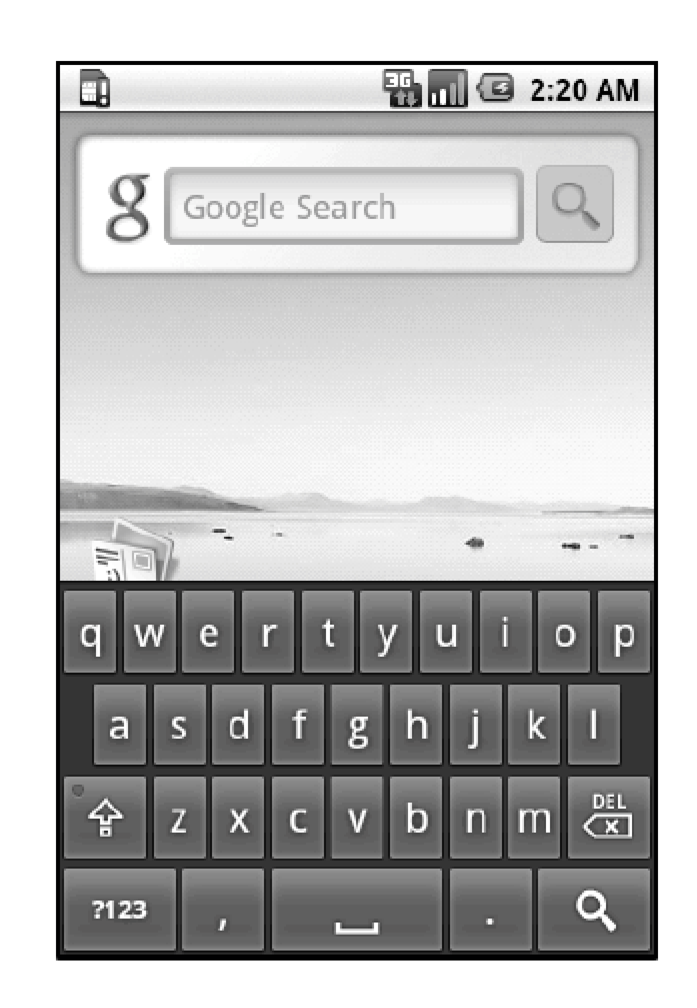



图 2.27　SoftKeyboard 示例

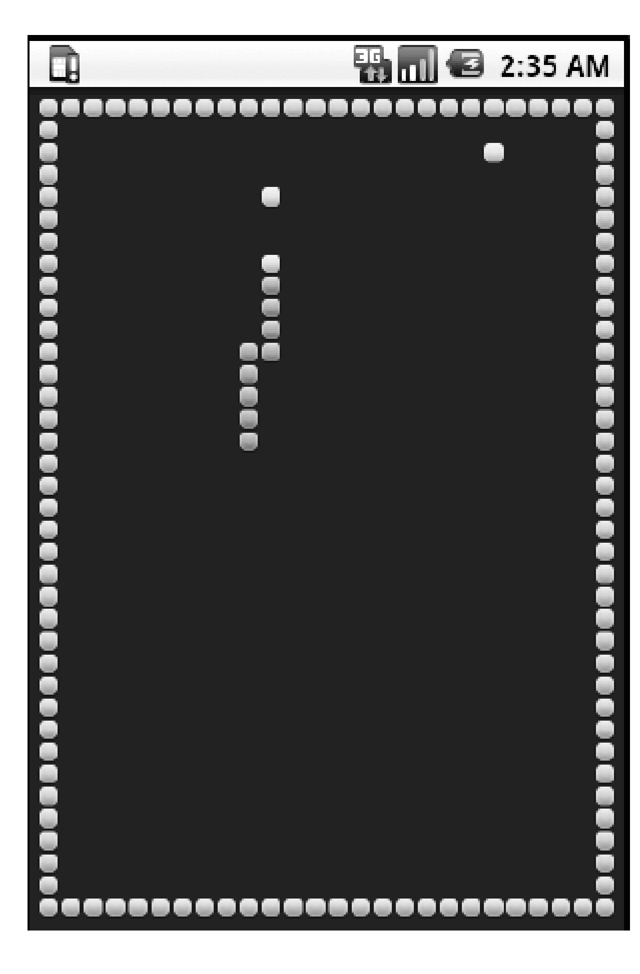



图 2.28　Snake 示例

8）LunarLander 示例

LunarLander 示例也是一个小游戏，模拟登陆舱在月球表面着陆。用户通过控制登陆舱的方向和速度，使登陆舱可以平稳地在月球表面着陆。LunarLander 示例实现了简单的碰撞检查功能，值得游戏开发人员学习和参考。LunarLander 示例如图 2.29 所示。

9）JetBoy 示例

JetBoy 示例是一个支持背景音乐和音效的游戏程序，用户可以控制飞船击碎飞来的陨石。JetBoy 示例如图 2.30 所示。




图 2.29　SoftKeyboard 示例

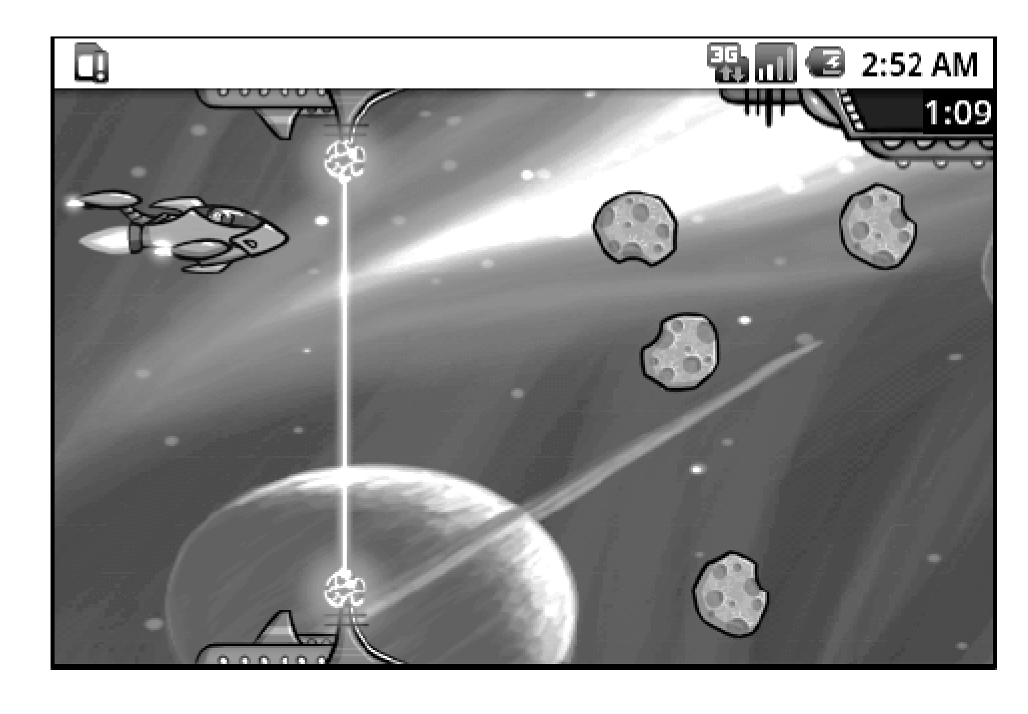



图 2.30　JetBoy 示例

## 223　开发工具

Android SDK 提供了多个强大的开发工具，便于程序开发人员简化开发和调试过程。这些工具中多数可以在 Eclipse 中直接调用，但也有部分工具需要在命令行模式下使用，下面的内容将逐个介绍这些工具。

### 1. Android 模拟器

Android SDK 中最重要的工具就是 Android 模拟器，模拟器文件的位置是<Android SDK>/tools/emulator.exe。正是因为 Android 模拟器的存在，让程序开发人员在没有实际设备的情况，可以对 Android 应用程序进行开发、调试和仿真。

Android 模拟器还支持加载 SD 卡映像文件，更改模拟网络状态，延迟和速度，模拟电话呼叫和接收短信等。但到目前为止，Android 模拟器仍有大量不支持的功能，包括接听真实电话呼叫，USB 连接，摄像头捕获，设备耳机，电池电量和 AC 电源检测，SD 卡插拔检查和使用蓝牙设备等。

Android 模拟器还支持多种外观和特性，包括屏幕的尺寸、方向等，可以参考表 2.1。

**表 2.1　Android 模拟器外观**

| 外　　观 | 外观 ID | 说　　明 |
|---|---|---|
| | HVGA-L | 解析度：480×320<br>方向：横向 |
| | HVGA-P | 解析度：320×480<br>方向：纵向<br>缺省配置 |
| | QVGA-L | 解析度：320×240<br>方向：横向 |
| | QVGA-P | 解析度：240×320<br>方向：纵向 |

Android 模拟器支持将屏幕当成触摸屏使用，可以使用鼠标单击屏幕模拟用户对 Android 设备的触摸操纵。同时，在 Android 模拟器上有普通手机设备常见的各种按键，如音量键、挂断键、返回键和菜单键等，按键的布局参见图 2.31。

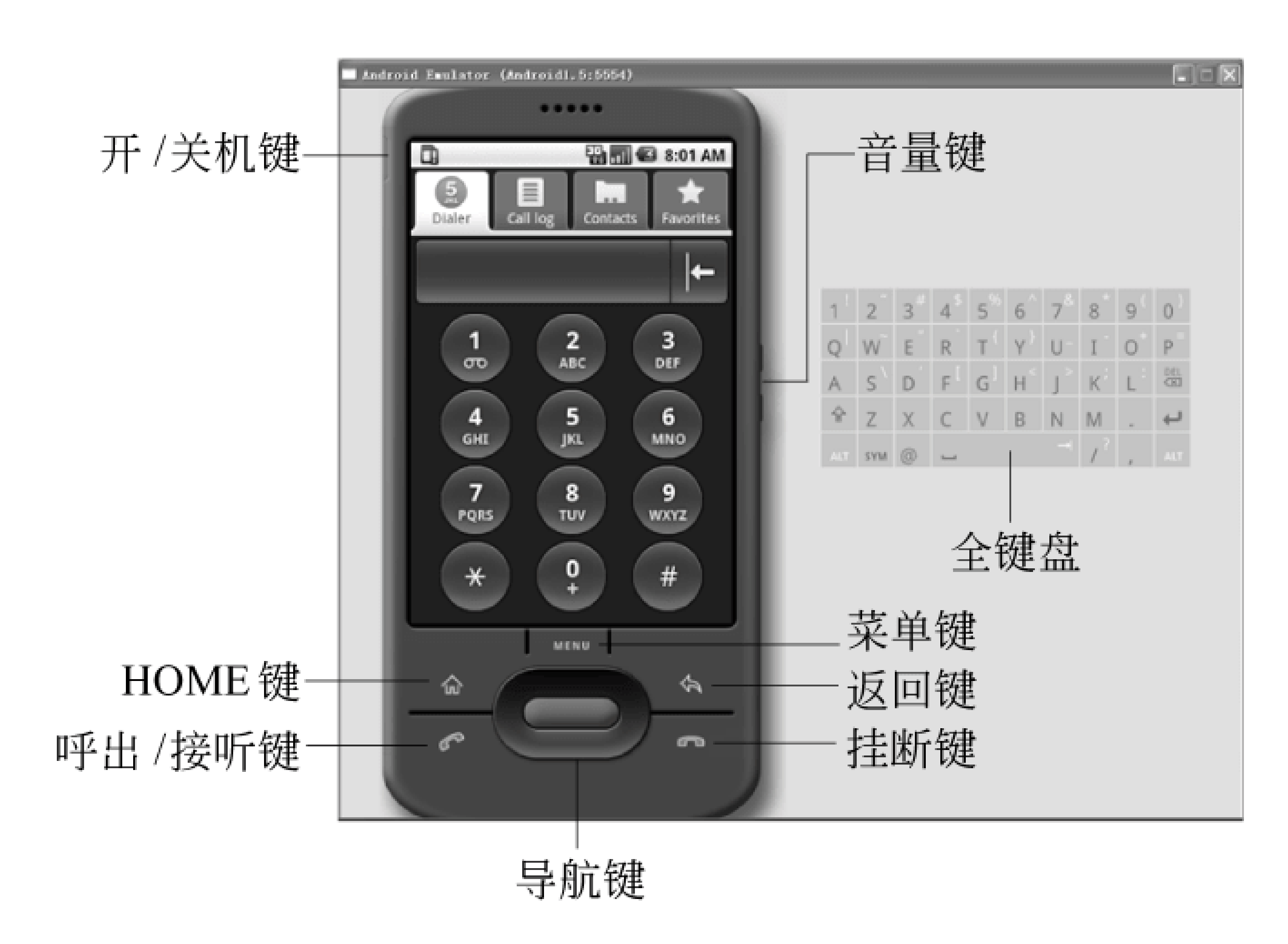



**图 2.31　Android 模拟器按键布局**

### 2. Android 调试桥

Android 调试桥(Android Debug Bridge，ADB)是用于连接 Android 设备和模拟器的工具，支持应用程序安装、调试和传输文件。Android 调试桥是一个客户端/服务器程序，包含守护程序、服务器程序和客户端程序。守护程序运行在每个模拟器的后台；服务器程序运行在开发环境中，管理客户端和守护程序的连接；客户端程序通过服务器程序，与模拟器中的守护程序相连接。

### 3. DDMS

DDMS(Dalvik Debug Monitor Service)是 Android 系统中内置了调试工具，可以用来监视 Android 系统中进程、堆栈信息，查看 logcat 日志，实现端口转发服务和屏幕截图功能，模拟器电话呼叫和 SMS 短信，以及浏览 Android 模拟器文件系统等。DDMS 的启动文件是＜Android SDK＞/tools/ddms.bat。

在 Eclipse 中，选择 Windows→Open Perspective→DDMS 命令，打开 DDMS 调试界面，然后选择 Windows→Show view→Other 命令，打开 Show View 的选择对话框，如图 2.32 所示。这样就可以在 DDMS 调试界面中添加任何希望进行调试和检查的功能。

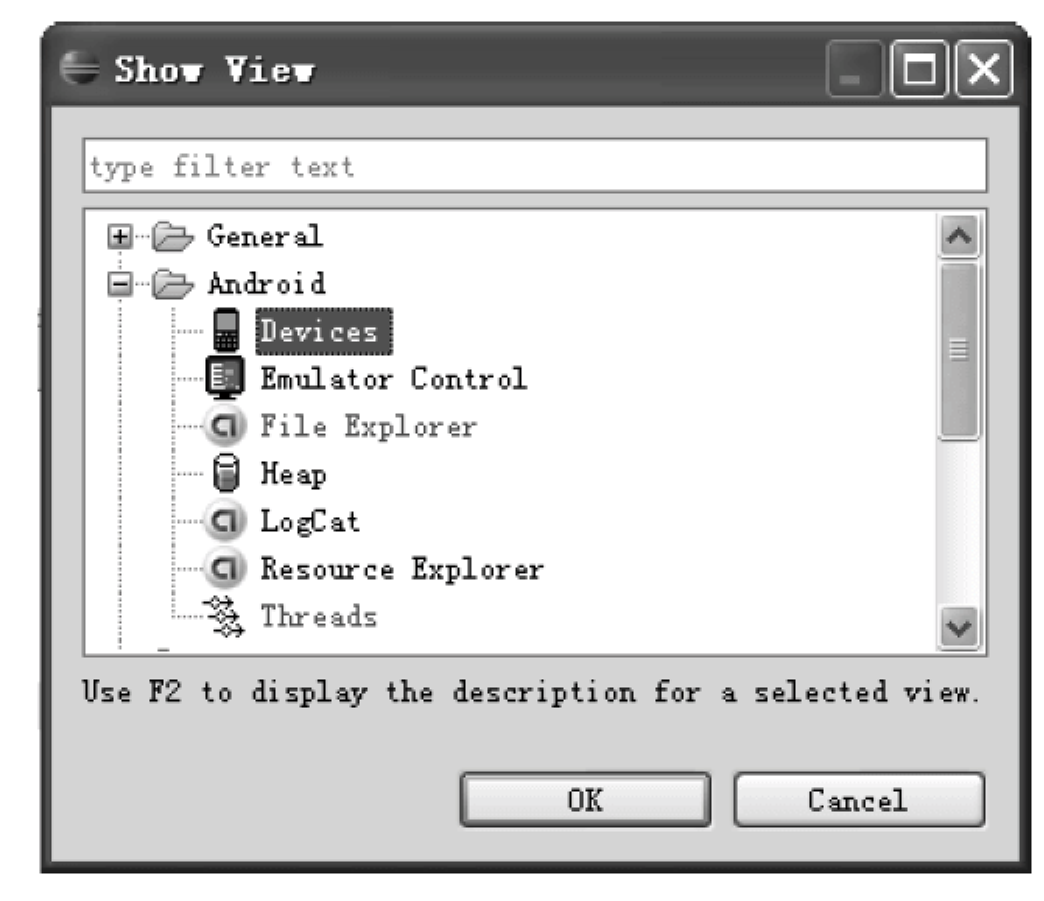



**图 2.32　Show View 选择对话框**

DDMS 中的设备管理器(Devices)，可以同时监控多个 Android 模拟器，显示每个模拟器中所有正在运行的进程。模拟器使用端口号进行唯一标识，例如监听端口是 5554 的模拟器则

标识为 emulator-5554。在选择指定的进程后，可以通过右上角的按钮刷新进程中线程和堆栈的信息，或是单击 STOP 按钮关闭指定进程。另外，还提供屏幕截图功能，可以将 Android 模拟器当前的屏幕内容保存成 png 文件。DDMS 中的设备管理器如图 2.33 所示。

Devices

| Name | | | |
|---|---|---|---|
| emulator-5554 | Online | | Android1.5... |
| system_process | 573 | | 8600 / 8700 |
| com.android.phone | 693 | | 8601 |
| android.process.acore | 708 | | 8602 |
| com.android.alarmclock | 712 | | 8603 |
| android.process.media | 714 | | 8604 |
| com.android.mms | 744 | | 8605 |
| com.android.inputmethod.latin | 763 | | 8608 |

**图 2.33　DDMS 中的设备管理器**

DDMS 中的模拟器控制器(Emulator Control)，可以控制 Android 模拟器的网络速度和延迟，模拟语音和 SMS 短信通信。模拟器控制支持的网络速率包括 GSM、HSCSD、PRS、EDGE、MTS、DPA 和全速率，支持的网络延迟有 GPRS、EDGE、UMTS 和无延迟。DDMS 中的模拟器控制器如图 2.34 所示。

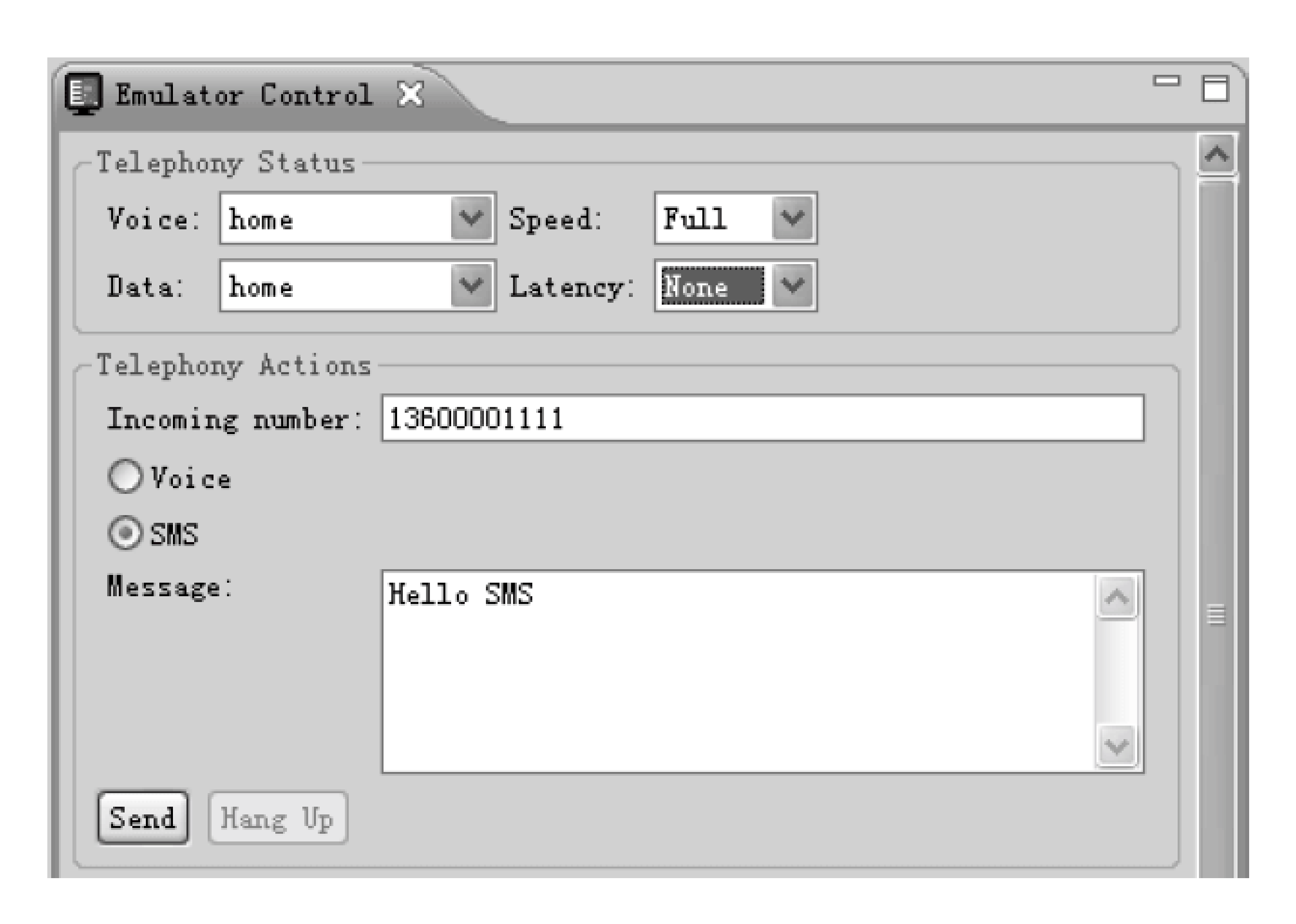
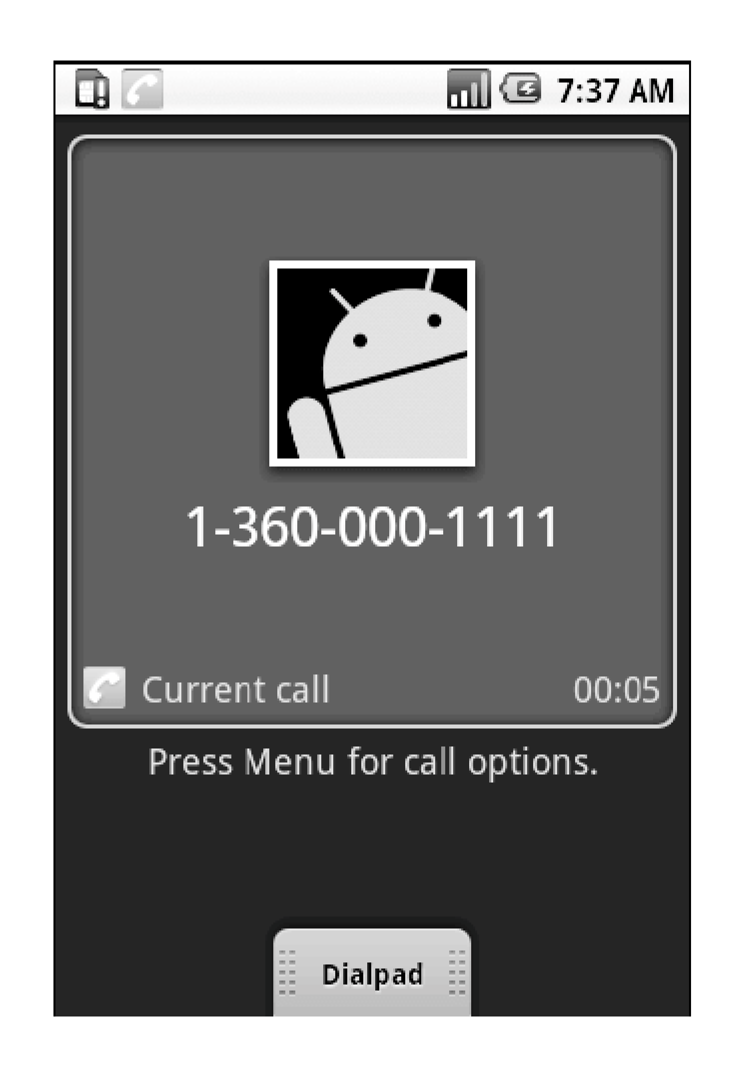



**图 2.34　DDMS 中的模拟器控制器**

在 Telephony Actions 中的 Incoming number 文本框中输入呼叫的电话号码，然后选择语言呼叫(Voice)单选项，单击 Send 按钮后，模拟器就可以接收到来自输入电话号码的语音电话，如图 2.35 左图所示。如果选择短信(SMS)，在 Message 中填入短信的内容，模拟器就可以接收到来自输入电话号码的 SMS 短信，如图 2.35 右图所示。

DDMS 中的文件浏览器(File Explorer)，可以对 Android 内置存储器上的文件进行上传、下载和删除等操作，还可以显示文件和目录的名称、权限、建立时间等信息。DDMS 中的文件浏览器制器如图 2.36 所示。

DDMS 中的日志浏览器(LogCat)，可以浏览 Android 系统、Dalvik 虚拟机或应用程序产生的日志信息，有助于快速定位应用程序产生的错误。DDMS 中的日志浏览器如图 2.37 所示。



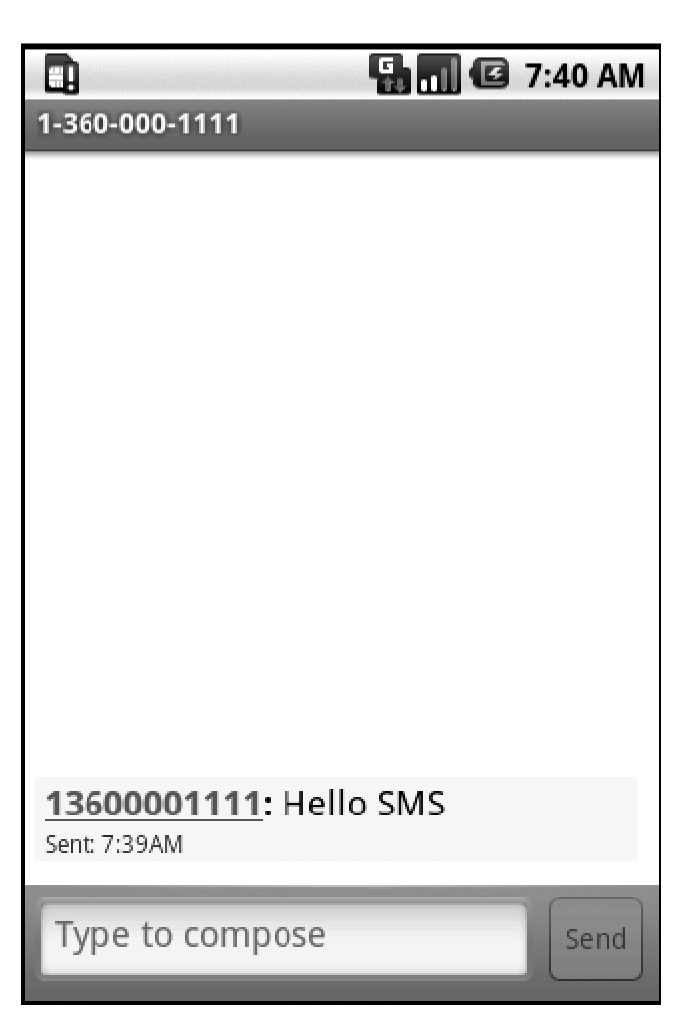
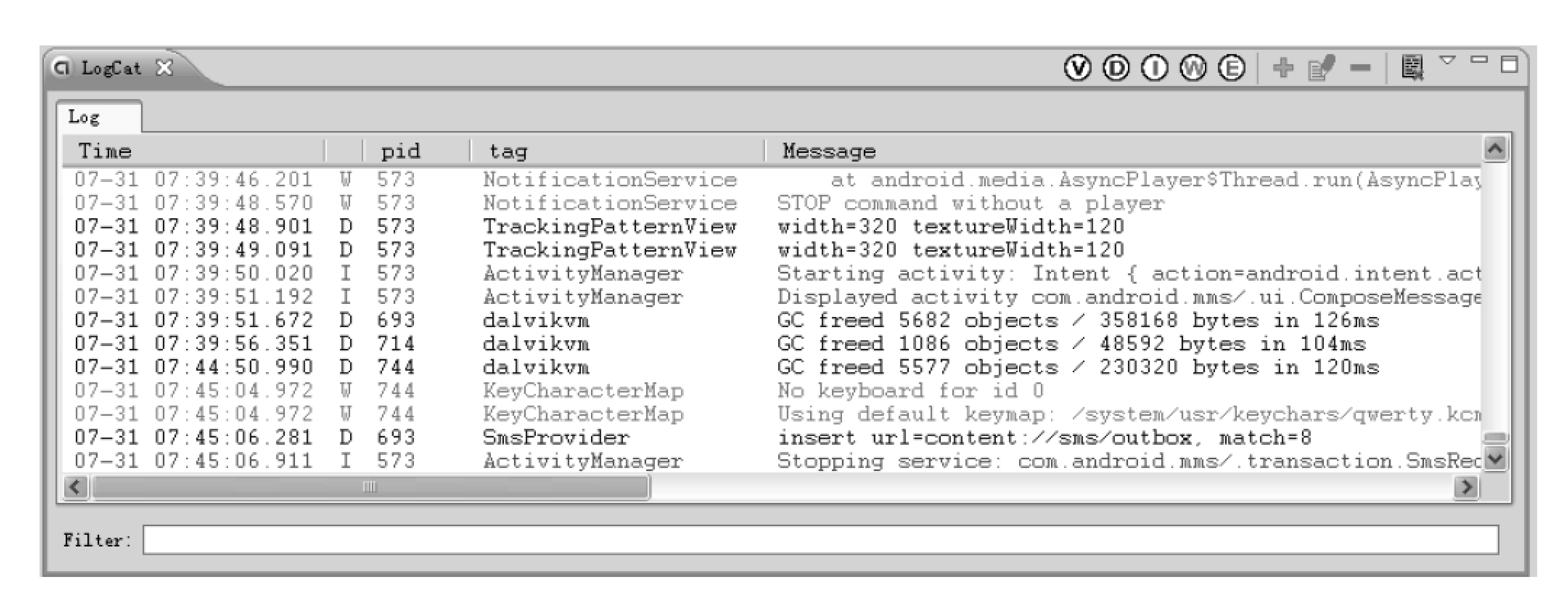




图 2.35 Android 电话呼叫和 SMS 短信

File Explorer

| Name | Size | Date | Time | Permiss... | Info |
|---|---|---|---|---|---|
| data | | 2009-06-20 | 00:53 | drwxrwx--x | |
| anr | | 2009-06-20 | 00:54 | drwxrwxrwx | |
| app | | 2009-06-20 | 00:53 | drwxrwx--x | |
| app-private | | 2009-06-20 | 00:53 | drwxrwx--x | |
| dalvik-cach | | 2009-06-20 | 00:53 | drwxrwx--x | |
| data | | 2009-06-20 | 00:53 | drwxrwx--x | |
| local | | 2009-06-20 | 00:53 | drwxrwx--x | |
| lost+found | | 2009-06-20 | 00:53 | drwxrwx--- | |
| misc | | 2009-06-20 | 00:53 | drwxrwx--t | |
| property | | 2009-06-20 | 00:53 | drwx------ | |
| system | | 2009-06-20 | 00:54 | drwxrwxr-x | |
| sdcard | | 2009-07-31 | 04:45 | d--------- | |
| system | | 2009-04-22 | 04:11 | drwxr-xr-x | |

图 2.36 DDMS 中的文件浏览器



LogCat

Log

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 07-31 07:39:46.201 | W | 573 | NotificationService | at android.media.AsyncPlayer$Thread.run(AsyncPlay |
| 07-31 07:39:48.570 | W | 573 | NotificationService | STOP command without a player |
| 07-31 07:39:48.901 | D | 573 | TrackingPatternView | width=320 textureWidth=120 |
| 07-31 07:39:49.091 | D | 573 | TrackingPatternView | width=320 textureWidth=120 |
| 07-31 07:39:50.020 | I | 573 | ActivityManager | Starting activity: Intent { action=android.intent.act |
| 07-31 07:39:51.192 | I | 573 | ActivityManager | Displayed activity com.android.mms/.ui.ComposeMessage |
| 07-31 07:39:51.672 | D | 693 | dalvikvm | GC freed 5682 objects / 358168 bytes in 126ms |
| 07-31 07:39:56.351 | D | 714 | dalvikvm | GC freed 1086 objects / 48592 bytes in 104ms |
| 07-31 07:44:50.990 | D | 744 | dalvikvm | GC freed 5577 objects / 230320 bytes in 120ms |
| 07-31 07:45:04.972 | W | 744 | KeyCharacterMap | No keyboard for id 0 |
| 07-31 07:45:04.972 | W | 744 | KeyCharacterMap | Using default keymap: /system/usr/keychars/qwerty.kcm |
| 07-31 07:45:06.281 | D | 693 | SmsProvider | insert url=content://sms/outbox, match=8 |
| 07-31 07:45:06.911 | I | 573 | ActivityManager | Stopping service: com.android.mms/.transaction.SmsRec |

Filter:

图 2.37 DDMS 中的日志浏览器

除了上面介绍的功能外，DDMS还能够查看虚拟机的堆栈状态、线程信息和控制台信息。由此可见，DDMS是进程程序调试和错误定位的强大工具。

### 4. 其他工具

为了便于Android程序开发，Android SDK还提供了一些小工具。这些工具的名

称、用途和启动文件如表 2.2 所示。

**表 2.2 Android SDK 提供的其他工具**

| 工具名称 | 启动文件 | 说明 |
| --- | --- | --- |
| 数据库工具 | sqlite3.exe | 用来创建和管理 SQLite 数据库 |
| 打包工具 | apkbuilder.bat | 将应用程序打包成 apk 文件 |
| 层级观察器 | hierarchyviewer.bat | 对用户界面进行分析和调试,以图形化的方式展示树形结构的界面布局 |
| 跟踪显示工具 | traceview.bat | 以图形化的方式显示应用程序的执行日志,用来调试应用程序,分析执行效率 |
| SD 卡映像创建工具 | mksdcard.exe | 建立 SD 卡的映像文件 |
| NinePatch 文件编辑工具 | draw9patch.bat | NinePatch 是 Android 提供的可伸缩的图形文件格式,基于 PNG 文件。draw9patch 工具可以使用所见即所得编辑器建立 NinePatch 文件 |

## 习　　题

1. 尝试安装 Android 开发环境,并记录安装和配置过程中所遇到的问题。

2. 浏览 Android SDK 帮助文档,了解 Android SDK 帮助文档的结构和用途,这样会对以后的学习带来极大的便利。

3. 在 Android SDK 中,Android 模拟器、Android 调试桥和 DDMS 是 Android 应用程序开发过程中经常使用到的 3 个工具,简述这 3 个工具的用途。

4. 为了进一步熟悉 Android 模拟器,通过命令行方式启动模拟器,并在模拟器中尝试使用各种功能和应用软件。命令行方式启动模拟器的方法是在＜Android SDK＞/tools 目录中,输入命令 emulator-data test 即可。

# 第 3 章 chapter 3

# Android 应用程序

本章主要介绍了开发 Android 应用程序的基础知识和基本方法。通过本章内容的学习，读者可以完全掌握使用 Eclipse 开发 Android 应用程序的过程和方法，了解 Android 应用程序的目录结构和自动生成文件的作用。学习使用命令行创建 Android 应用程序，有助于深入理解 Android 程序的生成、安装和运行过程。

**本章学习目标：**

- 掌握使用 Eclipse 开发 Android 应用程序的方法
- 掌握 Android 虚拟设备(AVD)的创建方法
- 了解 R. java 文件的用途和生成方法
- 了解 AndroidManifest. xml 文件的用途
- 了解 Android 的程序结构
- 了解使用命令行创建 Android 应用程序方法

## 3.1 第一个 Android 程序

本小节将介绍如何使用 Eclipse 集成开发环境建立第一个 Android 程序 HelloAndroid。首先启动 Eclipse，显示 Eclipse 集成开发环境如图 3. 1 所示。如果在 Eclipse 中建立过 Android 工程，工程名称和目录结构将显示在 Package Explorer 区域内。

有两种方法可以打开 Android 工程向导，一种是以选择 File→New→Project…| Android→Android Project 的顺序；另一种是以选择 File→New→Other…| Android→Android Project 的顺序。两种方法只是选择的顺序不同，结果是相同的，都可以打开 Android 工程向导，如图 3. 2 所示。在第二种方法中，除了可以建立 Android 工程向导外，还可以建立 Android XML 文件，用于建立用户界面或存储数值等用途。

在 Android 工程向导中，需要确定工程名称、保存位置、SDK 版本、应用程序名称、包名称和 SDK 最低版本等信息，后续内容将对每一项进行详细介绍。第一步需要填入工程名称(Project Name)，工程名称必须唯一，不能与已有工程重名，填入 HelloAndroid 作为工程名称。

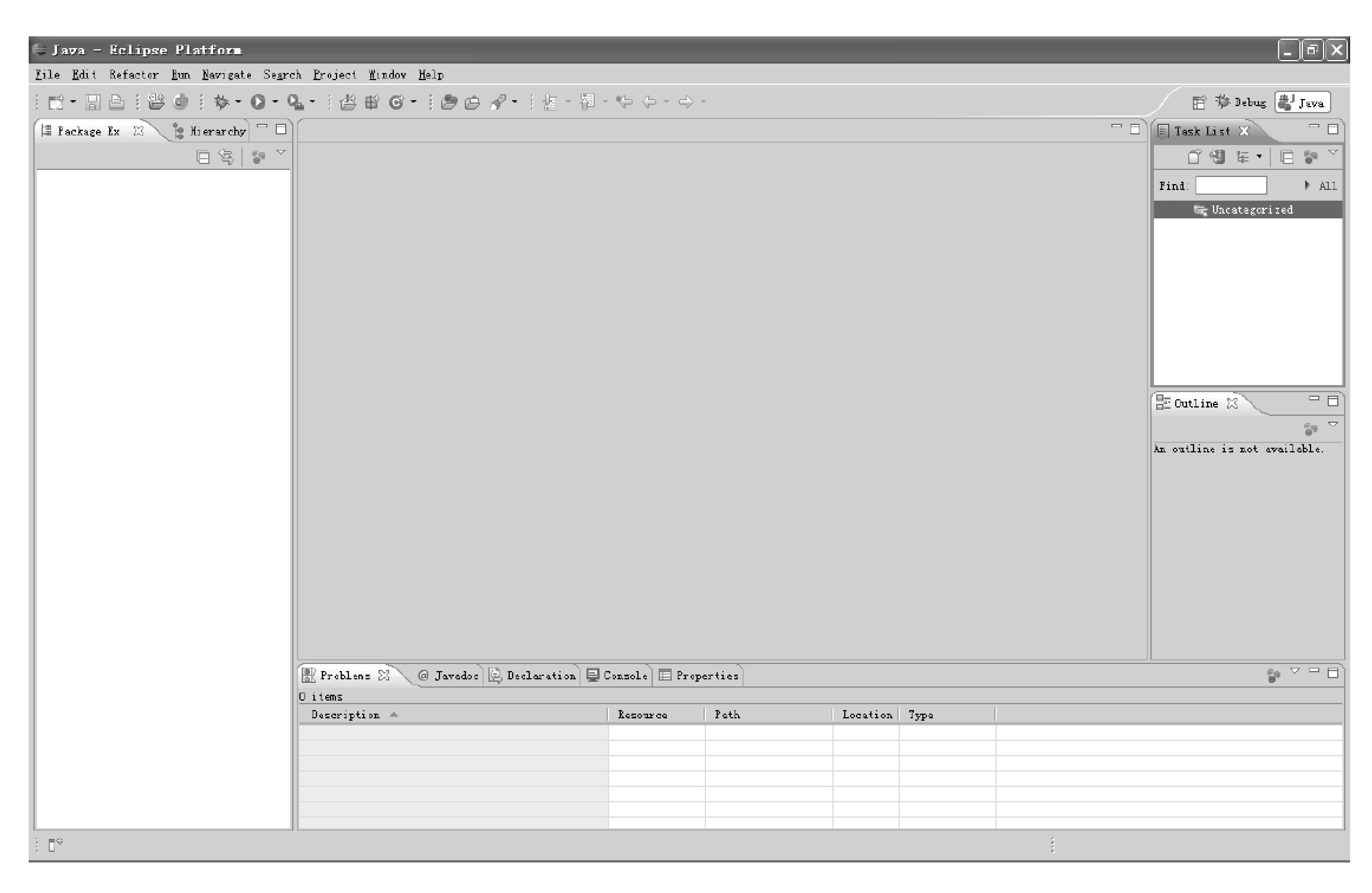



图 3.1 Eclipse 集成开发环境

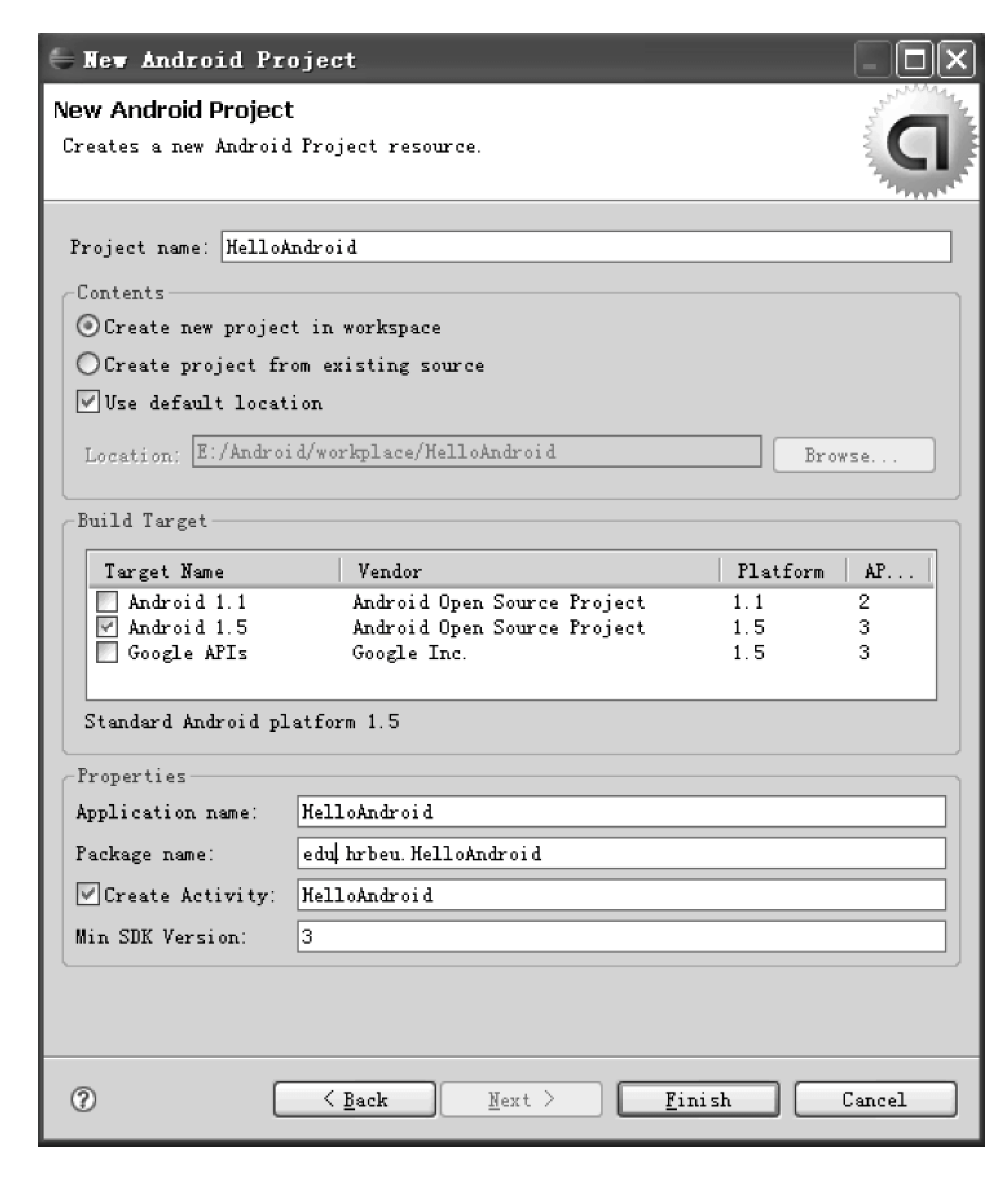



图 3.2 Android 工程向导

第二步是在 Contents 中选择 Android 工程的创建方式，默认是在工作空间中创建新 Android 工程，也可以选择利用已有代码创建 Android 工程，读者可以不做任何更改，保持默认设置。当然，也可以取消选中 Use default location 复选框，选择其他位置保存 Android 工程。作者的 Eclipse 工作空间是 E:/Android/workplace，因此 HelloAndroid 工程的保存位置在 E:/Android/workplace/HelloAndroid。

第三步是在 Build Target 中选择编译目标，即 Android 程序所运行的 Android 系统版本和是否支持附加库。其中有三个复选框，第一个是标准的 1.1 版本 Android 系统，API 等级为 2；第二个是标准的 1.5 版本 Android 系统，API 等级为 3；第三个是包含 Google API 的 1.5 版本 Android 系统，API 等级为 3。选择 Android 程序的编译目标，应该根据目标手机所支持的 Android 系统版本所定。选择不同版本的 Android 系统，在工程中将引入不同版本的 android.jar 包；选择支持 Google API 的 Android 系统，将引入支持 Google 地图功能的 maps.jar 包。选择第二个复选框，标准的 1.5 版本 Android 系统。

第四步是在 Properties 中填入相关信息。应用程序名称（Application Name）是 Android 程序在手机中显示的名称，程序运行时也会显示在手机的顶部。给第一个 Android 程序起一个英文名称 HelloAndroid，当然也可以起中文的应用程序名称，例如“第一个 Android 程序”等。

包名称（Package Name）是包的命名空间，需要遵循 Java 包的命名方法。包名称由两个或多个标识符组成，中间用点隔开，例如 hrbeu.HelloAndroid。使用包主要为了避免命名冲突，因此可以使用反写电子邮件地址的方式保证命名的唯一性，例如笔者的电子邮件地址是 wangxianghui@hrbeu.edu.cn，则可以将包名称命名为 cn.edu.hrbeu.wangxianghui。为了保证代码的简洁，第一个 Android 程序的包名称使用 edu.hrbeu.HelloAndroid。

创建 Activity(Create Activity)是一个可选项，如果需要自动生成一个 Activity 的代码文件，则需要选择该复选框，否则可以不选。Create Activity 主要用于管理用户界面，后续章节将做详细介绍，选择该复选框并将 Create Activity 的名称定为 HelloAndroid。Create Activity 的名称完全可以与应用程序名称不同，但一般为了简洁明了，会使这个 Create Activity 的名称与应用程序的名称相同，表示这个 Create Activity 是 Android 程序运行时首先显示给用户的界面。

SDK 最低版本（Min SDK Version）指的是 Android 程序能够运行的最低的 API 等级，如果手机中的 Android 系统的 API 等级低于程序的 SDK 最低版本，则程序不能在该 Android 系统中运行。在选择 Build Target 时，SDK 最低版本已被自动填入 3，此项无需更改。

最后单击 Finish 按钮，工程向导会根据用户所填写的 Android 工程信息，自动在后台创建 Android 工程所需要的基础文件和目录结构。当创建过程结束，用户将看到图 3.3 所显示的内容。

用户无需在 HelloAndroid 工程中添加任何代码，即可运行 HelloAndroid 程序。但为了让 Android 程序能够正常运行，必须首先建立 Android 虚拟设备（Android Virtual Device，AVD）。AVD 是对 Android 模拟器进行自定义的配置清单，能够配置 Android

模拟器的硬件列表、外观、Android 版本、SDK 库和存储配置等信息。因为在 1.5 版本的 Android SDK 中没有附带任何配置好的 AVD，因此即使是最简单的 HelloAndroid 程序，也需要建立一个 AVD，AVD 选择标准的 1.5 版本 Android 系统。

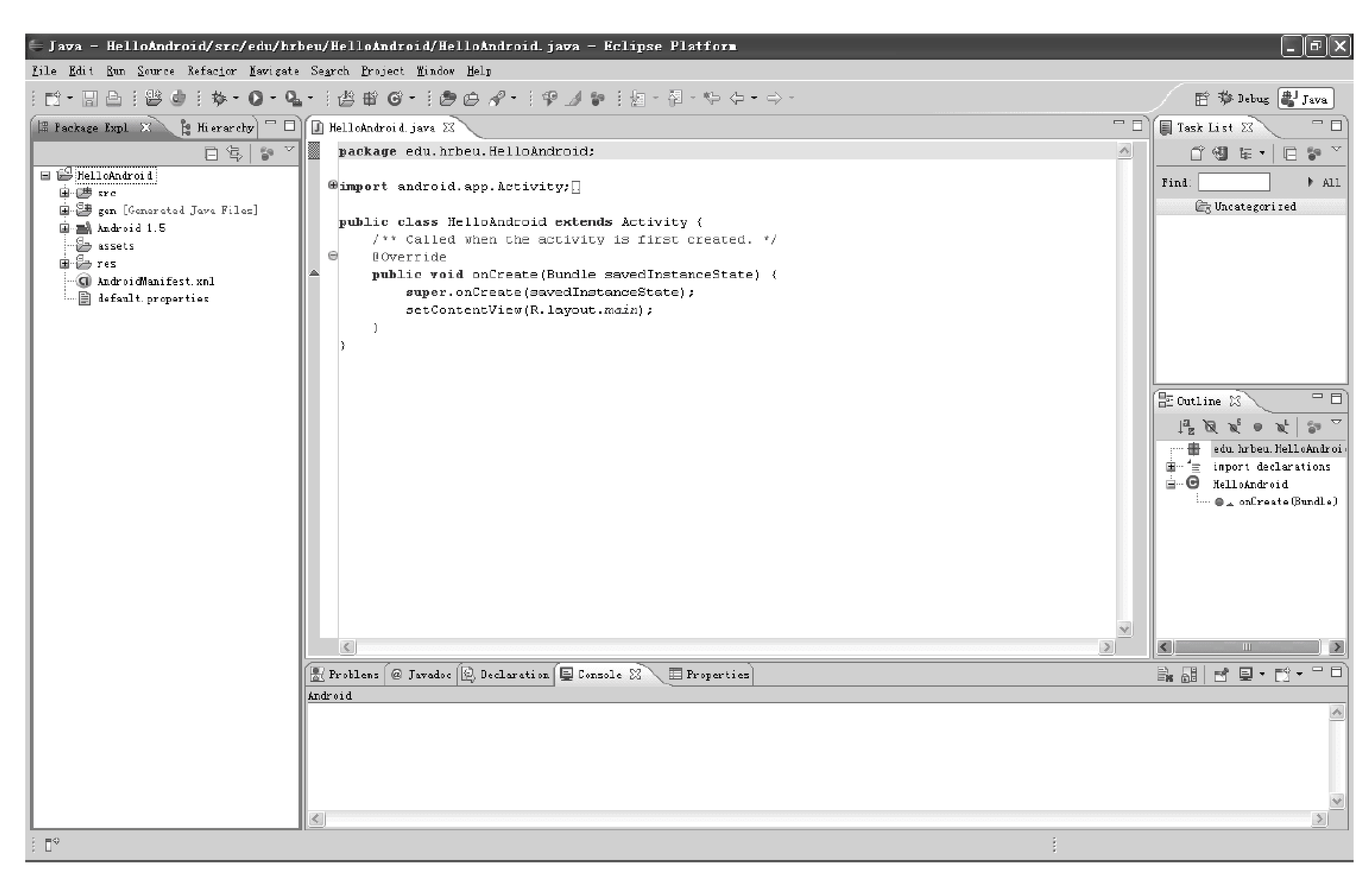



**图 3.3　HelloAndroid 工程的基础文件和目录结构**

建立 AVD 需要使用 Windows 系统的命令行工具 CMD，选择“开始”→“运行”→CMD 命令，启动命令行工具，并进入＜Android SDK＞/tools 目录下，其中＜Android SDK＞表示 Android SDK 所在的目录，本书的 Android SDK 安装在 E:\Android\android-sdk-windows-1.5_r1。

为了选择适合的编译目标，需要利用 android list targets 命令搜索＜Android SDK＞/platforms 和＜Android SDK＞/add-ons 目录下所有有效的 Android 系统映像，并将 Android 系统映像列表显示在命令行工具中，如图 3.4 所示。共有 3 个可以选择的编译目标，分别是 1.1 版本 Android 系统、1.5 版本 Android 系统和包含 Google API 的 1.5 版本 Android 系统。其中前两个系统映像的 Type 属性是 Platform，因此是标准的 Android 系统，使用者可自行配置模拟器的硬件清单；最后一个系统映像的 Type 属性是 Add-On，是一种定制系统，这种定制系统不能更改模拟器的硬件配置清单。

使用 android create avd -n android1.5 -t 2 命令，以 id 为 2 的 1.5 版本 Android 系统为目标，建立一个名为 Android1.5 的 AVD。其中，-n 参数表明 AVD 的名称，-t 参数表明选择的 Android 系统映像的 id 值。命名运行时会询问用户是否需要定制硬件配置清单(Do you wish to create a custom hardware profile)，选择 no，使用默认的硬件配置，运行结果如图 3.5 所示。当然也可以选择 yes，根据需要重新定制模拟器支持的硬件清单。

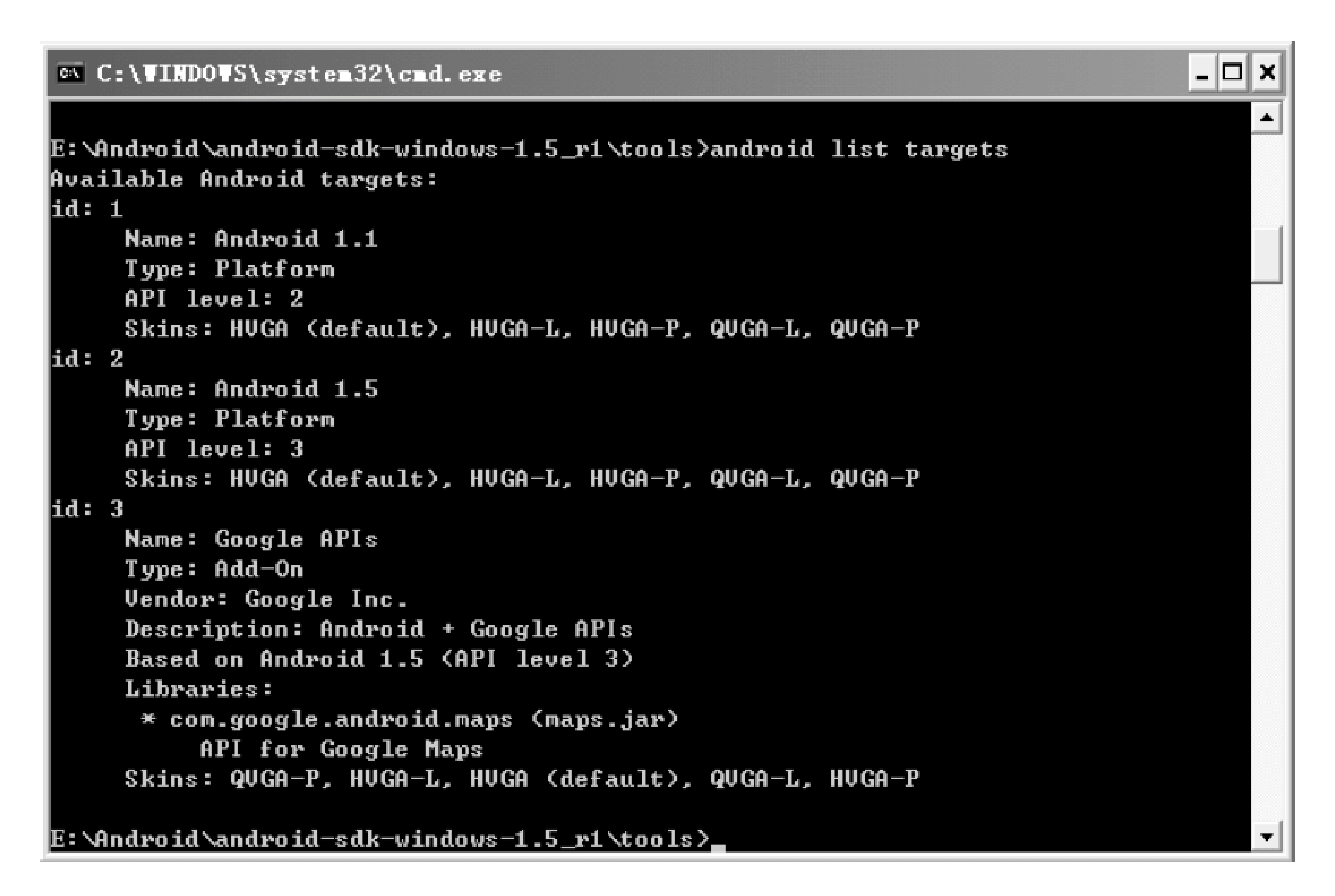



图 3.4 Android 系统映像列表

```
C:\WINDOWS\system32\cmd.exe
E:\Android\android-sdk-windows-1.5_r1\tools>android create avd -n android1.5 -t
2
Android 1.5 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
Created AVD 'android1.5' based on Android 1.5

E:\Android\android-sdk-windows-1.5_r1\tools>
```

图 3.5 建立 AVD

建立 AVD 过程中，Android 工具会在文件系统中建立 Android1.5.ini 文件和 Android1.5.avd 目录。Android1.5.ini 文件用来保存 Android1.5.avd 目录所在的位置；Android1.5.avd 目录则用以保存 AVD 配置文件、用户数据文件、SD 卡映像和其他模拟器运行过程中可能产生文件。Android1.5.ini 文件和 Android1.5.avd 目录的保存位置，会根据用户使用的操作系统不同而不同，如果用户使用的是 Windows XP 系统，则目录会保存在 C:\Documents and Settings\<user>\.android\下；如果用户使用的是 Windows Vista 系统，则目录则会保存在 C:\Users\<user>\.android 下。AVD 的管理命令和虚拟硬件列表可以参考附录 A。

HelloAndroid 程序正式运行前，还需要选择 Run→Run Configuration 命令或选择 Run→Debug Configuration 命令，配置模拟器的启动选项。图 3.6 是 Run Configuration 的启动选项，可以选择不同的 AVD、配置网络速度、网络延迟、控制台的字符编码和标准输入输出等内容。一般情况下，只要正确选择 AVD 即可，其他选项可以不做任何修改。

使用 Eclipse 运行 Android 程序非常简单，只要选择 Run→Run | Android Application 命令或选择 Run→Debug | Android Application 命令便可以启动模拟器，并运行 Android 程序。启动 Android 模拟器是一个缓慢的过程，程序调试完毕后，不必关闭 Android 模拟器，再次运行 Android 程序时，可以节约启动模拟器的时间。Eclipse 会自动完成 Android 程序编译、打包和上传等过程，并将程序的运行结果显示在模拟器中。HelloAndroid 程序的运行结果如图 3.7 所示。

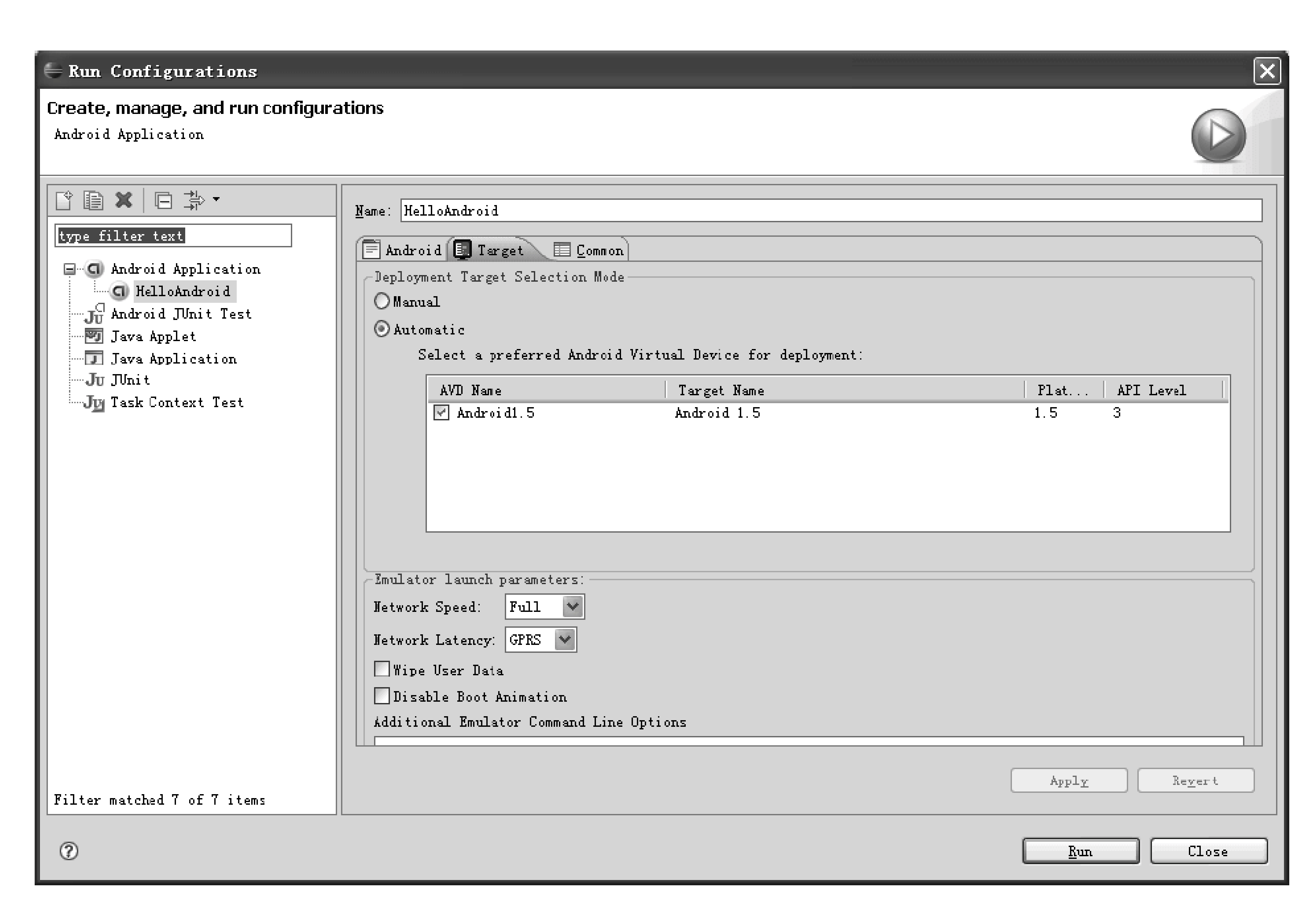



图 3.6 Run Configuration 启动选项




图 3.7 HelloAndroid 的运行结果

目前,已经完成了第一个 Android 程序,并得到了程序的运行结果,对如何建立和运行 Android 程序已经有了基本的了解。3.2 节仍然以 HelloAndroid 为例,介绍 Android 的程序目录结构和文件用途。

## 3.2 Android 程序结构

在建立 HelloAndroid 程序的过程中,ADT 插件会自动建立一些目录和文件,如图 3.8 所示。这些目录和文件有其固定的作用,有的允许修改,有的则不能进行修改,了解这些文件和目录,对 Android 程序开发有着非常重要的作用。

在 Package Explore 中，ADT 以工程名称 HelloAndroid 作为根目录，将所有自动生成的和非自动生成的文件都保存在这个根目录下。根目录下包含 4 个子目录 src、assets、res 和 gen，一个库文件 android. jar，以及两个工程文件 Androidmanifest. xml 和 default. properties。

src 目录是源代码目录，所有允许用户修改的 Java 文件和用户自己添加的 Java 文件都保存在这个目录中。HelloAndroid 工程建立初期，ADT 根据用户在工程向导中的 Create Activity 选项，自动建立 HelloAndroid. java 文件。

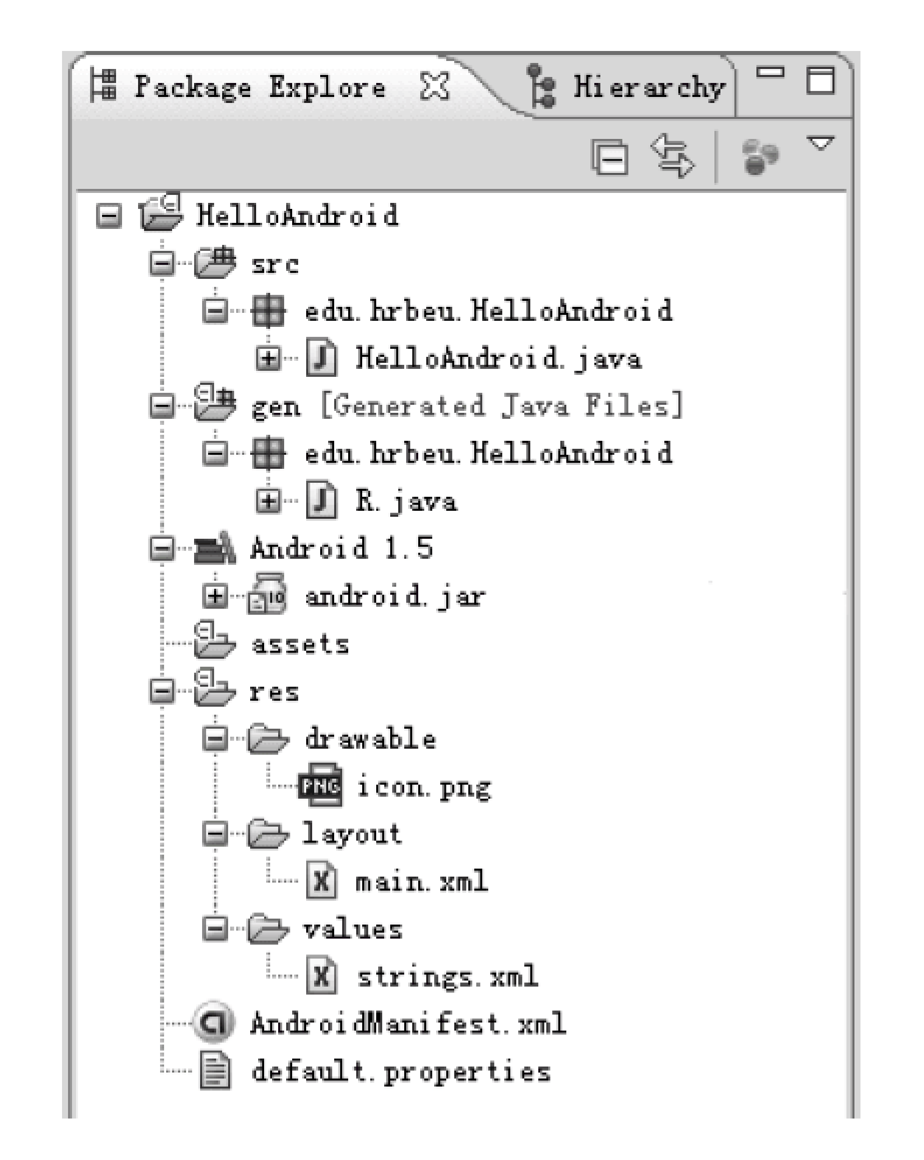



**图 3.8 HelloAndroid 工程的目录和文件**

assets 目录用来存放原始格式的文件，例如音频文件、视频文件等二进制格式文件。此目录中的资源不能够被 R. java 文件索引，因此只能以字节流的形式进行读取。一般情况下为空。

res 目录是资源目录，Android 程序所有的图像、颜色、风格、主题、界面布局和字符串等资源都保存在其下的三个子目录中。其中，drawable 目录用来保存图像文件，layout 目录用来保存与用户界面相关的布局文件，values 目录保存颜色、风格、主题和字符串等资源。在 HelloAndroid 工程中，ADT 在 drawable 目录中自动引入了 icon. png 文件，作为 HelloAndroid 程序的图标文件；在 layout 目录生成了 mail. xml 文件，用以描述图 3.7 所显示的用户界面；在 values 目录生成了 strings. xml 文件，将应用程序名称“HelloAndroid”和界面显示的“Hello World，HelloAndroid！”保存在这个文件中。

gen 目录是 1.5 版本新增的目录，用来保存 ADT 自动生成的 Java 文件，例如 R. java 或 AIDL 文件。

android. jar 文件是 Android 程序所能引用的函数库文件，Android 通用平台所支持 API 都包含在这个文件中，具体内容参考附录 B。

default. properties 文件记录了 Android 工程的相关设置，例如编译目标和 apk 设置等，该文件不能手工修改，如果需要更改其中的设置，必须通过右键单击工程名称，在弹出的快捷菜单中选择 Properties 选项修改。从 default. properties 文件的代码中可以发现，大部分都是内容注释，仅有第 12 行是有效代码，说明了 Android 程序的编译目标。

default. properties 文件的代码如下：

```
1    # This file is automatically generated by Android Tools.
2    # Do not modify this file -- YOUR CHANGES WILL BE ERASED!
3    #
4    # This file must be checked in Version Control Systems.
5    #
6    # To customize properties used by the Ant build system use,
```

```
# "build.properties", and override values to adapt the script to your
# project structure.

# Project target.
target=android-3
```

AndroidManifest.xml 是 XML 格式的 Android 程序声明文件，包含了 Android 系统运行 Android 程序前所必须掌握的重要信息，这些信息包括应用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等，而且每个 Android 程序必须在根目录下包含一个 AndroidManifest.xml 文件。AndroidManifest 文件的具体内容可以参考附录 D。

XML 是一种可扩展标记语言，本身独立于任何编程语言，能够对复杂的数据进行编码，且易于理解。Android 工程中多处使用了 XML 文件，使应用程序开发更加具有弹性，且易于后期的维护和理解。

AndroidManifest.xml 文件的代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.hrbeu.HelloAndroid"
      android:versionCode="1"
      android:versionName="1.0">
  <application android:icon="@drawable/icon"
                    android:label="@string/app_name">
      <activity android:name=".HelloAndroid"
                    android:label="@string/app_name">
            <intent-filter>
                  <action android:name="android.intent.action.MAIN"/>
                  <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
      </activity>
  </application>
  <uses-sdk android:minSdkVersion="3"/>
</manifest>
```

在 AndroidManifest.xml 文件中，根元素是 manifest，包含了 xmlns:android、package、android:versionCode 和 android:versionName 等 4 个属性。xmlns:android 定义了 Android 的命名空间，值为 http://schemas.android.com/apk/res/android；package 定义了应用程序的包名称；android:versionCode 定义了应用程序的版本号，是一个整数值，数值越大说明版本越新，但仅在程序内部使用，并不提供给应用程序的使用者；android:versionName 定义了应用程序的版本名称，是一个字符串，仅限于为用户提供一个版本标识。

manifest 元素仅能包含一个 application 元素，application 元素中能够声明 Android 程序中最重要的 4 个组成部分，包括 Activity、Service、BroadcastReceiver 和

ContentProvider，所定义的属性将影响所有组成部分。第 6 行属性 android:icon 定义了 Android 应用程序的图标，其中@drawable/icon 是一种资源引用方式，表示资源类型是图像，资源名称为 icon，对应的资源文件为 res/drawable 目录下的 icon. png；第 7 行属性 android:label 则定义了 Android 应用程序的标签名称。

activity 元素是对 Activity 子类的声明，不在 AndroidManifest. xml 文件中声明的 Activity 将不能在用户界面中显示。第 8 行属性 android:name 定义了实现 Activity 类的名称，可以是完整的类名称，如 edu. hrbeu. HelloAndroid，也可以是简化后的类名称，如 . HelloAndroid；第 9 行属性 android:label 则定义了 Activity 的标签名称，标签名称将在用户界面的 Activity 上部显示，@string/app_name 同样属于资源引用，表示资源类型是字符串，资源名称为 app_name，资源保存在 res/values 目录下的 strings. xml 文件中。

intent-filter 中声明了两个子元素 action 和 category，在这里不详细讨论两个子元素的用途，但可以肯定的是，intent-filter 使 HelloAndroid 程序在启动时，将. HelloAndroid 这个 Activity 作为默认启动模块。

DT 包含了一个可视化的编辑器，如图 3. 9 所示，双击 AndroidManifest. xml 文件可直接进入可视化编辑器。用户可以在不接触 XML 的情况下，编辑 Android 工程的应用程序名称、包名称、图标、标签和许可等相关属性。

**图 3. 9　HelloAndroid 工程的目录和文件**

AR. java 文件是 ADT 自动生成的文件，包含对 drawable、layout 和 values 目录内的资源的引用指针，Android 程序能够直接通过 R 类引用目录中的资源。R. java 文件不能手工修改，所有代码必须由 ADT 自动生成。如果向资源目录中增加或删除了资源文件，则需要在工程名称上右击，在弹出的快捷菜单中选择 Refresh 选项来更新 R. java 文件中的代码。

HelloAndroid 工程生成的 R. java 文件的代码如下：

```
package edu.hrbeu.HelloAndroid;

```

```java
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon= 0x7f020000;
    }
    public static final class layout {
        public static final int main= 0x7f030000;
    }
    public static final class string {
        public static final int app_name= 0x7f040001;
        public static final int hello= 0x7f040000;
    }
}
```

R 类包含的几个内部类，分别与资源类型相对应，资源 ID 便保存在这些内部类中，例如子类 drawable 表示图像资源，内部的静态变量 icon 表示资源名称，其资源 ID 为 0x7f020000。一般情况下，资源名称与资源文件名相同（不包含扩展名），如 icon 对应 src/drawable 目录下的 icon. png 文件，main 对应 src/layout 目录下的 main. xml 文件。

资源的引用一般有两种情况，一种是在代码中引用资源，另一种则是在资源中引用资源。

在代码中引用资源时，需要在代码中使用资源 ID，可以通过[R. resource_type. resource_name]或者[andrlid. R. resource_type. resource_name]获取资源 ID。其中，resource_type 代表资源类型，也就是 R 类中的内部类名称；resource_name 代表资源名称，对应资源的文件名（不包含扩展名）或在 XML 文件中定义的资源名称属性。例如在 HelloAndroid. java 中，第 11 行代码便是在代码中对资源的引用。

在资源中引用资源时，一般的引用格式为@[package：]type：name。其中，@表示对资源的引用；package 是包名，如果在相同的包内，package 则可省略；type 是资源的类型，例如 string 或 drawable；name 是资源的名称。例如在 main. xml 文件中，第 10 行代码就是在资源中对资源的引用。

main. xml 文件的代码如下：

```xml
1  <?xml version="1.0"encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      >
7  <TextView
8      android:layout_width="fill_parent"
9      android:layout_height="wrap_content"
10     android:text="@string/hello"
```

```
11         />
12    </LinearLayout>
```

main. xml 文件是界面布局文件，利用 XML 语言描述的用户界面，界面布局的相关内容将在第 5 章用户界面设计中进行详细介绍。main. xml 代码的第 7 行说明在界面中使用 TextView 控件，TextView 控件主要用来显示字符串文本。代码第 10 行说明 TextView 控件需要显示的字符串，非常明显，@string/hello 是对资源的引用。通过 strings. xml 文件的第 3 行代码分析，在 TextView 控件中显示的字符串应是“Hello World，HelloAndroid!”。如果读者修改 strings. xml 文件的第 3 行代码的内容，重新编译、运行后，模拟器中显示的结果也应该随之更改。

strings. xml 文件的代码如下：

```
1    <?xml version="1.0"encoding="utf-8"?>
2    <resources>
3         <string name="hello">Hello World, HelloAndroid!</string>
4         <string name="app_name">HelloAndroid</string>
5    </resources>
```

HelloAndroid. java 是 Android 工程向导根据 Activity 名称创建的 Java 文件，这个文件完全可以手工修改。为了在 Android 系统上显示图形界面，需要使用代码继承 Activity 类，并在 onCreate()函数中声明需要显示的内容。

HelloAndroid. java 文件的代码如下：

```
1    package edu.hrbeu.HelloAndroid;
2
3    import android.app.Activity;
4    import android.os.Bundle;
5
6    public class HelloAndroid extends Activity {
7         /** Called when the activity is first created. */
8         @Override
9         public void onCreate(Bundle savedInstanceState) {
10             super.onCreate(savedInstanceState);
11             setContentView(R.layout.main);
12        }
13   }
```

代码的第 3 行和第 4 行，通过 android. jar 从 Android SDK 中引入了 Activity 和 Bundle 两个重要的包，用于子类继承和信息传递；第 6 行声明 HelloAndroid 类继承 Activity 类；第 8 行表明需要重写 onCreate()函数；第 9 行的 onCreate()会在 Activity 首次启动时会被调用，为了便于理解，可以认为 onCreate()是 HelloAndroid 程序的主入口函数；第 10 行调用父类的 onCreate()函数，并将 savedInstanceState 传递给父类，savedInstanceState 是 Activity 的状态信息；第 11 行声明了需要显示的用户界面，此界面是用 XML 语言描述的界面布局，保存在 scr/ layout/main. xml 资源文件中。

在分析了 Android 程序的目录结构和文件的用途后，对 AndroidManifest.xml 文件、Java 代码文件、资源引用和 R.java 等内容有了初步的了解，3.3 节将着重介绍如何使用命令行工具创建 Android 程序，有助于深入理解 Android 程序的生成、安装和运行过程。

## 3.3 使用命令行工具创建程序

到目前为止，读者应该非常熟悉如何使用 Eclipse 建立小型 Android 应用程序，Eclipse 能够为使用者提供良好的编辑和调试环境，并能够自动完成程序的编译、apk 打包和上传安装等过程。但 Eclipse 并不是 Android 程序的唯一开发环境，IntelliJ IDEA 和文本编辑器同样也能够完成 Android 程序开发。Android SDK 中包含了所有 Android 程序开发过程中所需要的编译、调试、打包和上传工具，能够在非 Eclipse 环境下帮助用户完成 Android 程序开发。本节将尝试使用文本编辑器和命令行工具创建 Android 程序 HelloCommondline，如果读者对使用命令行工具开发 Android 程序并不感兴趣，完全可以跳过本节。

Android 命令行工具都保存在＜Android SDK＞/tools 目录下，这些命令行工具在 Android SDK 起着非常重要的作用，因为即使用户使用 Eclipse 开发 Android 程序，很多重要的功能也都是调用 Android 命令行工具完成的，例如建立工程、在模拟中运行程序等。

使用命令行工具开发 Android 程序一般分为如下几个步骤：

(1) 使用 android.bat 建立 HelloCommondline 工程所需的目录和文件。

(2) 使用 Apache Ant 对 HelloCommondline 工程进行编译和 apk 打包。

(3) 使用 adb.exe 将 HelloCommondline 工程上传到 Android 模拟器中。

第一步，使用 android.bat 建立 HelloCommondline 工程所需的目录和文件。android.bat 是一个批处理文件，可以用来建立和更新 Android 工程，同时也可以管理 AVD。android.bat 能够创建 Android 工程所需要的目录结构和文件，表 3.1 给出了 android.bat 建立和更新 Android 工程的命令和参数说明。

**表 3.1 Android 工程管理命令**

| 命令 | 参数 | 说明 | 备注 |
|---|---|---|---|
| android create project | -k＜package＞ | 包名称 | 必备参数 |
| | -n＜name＞ | 工程名称 | |
| | -a＜activity＞ | Activity 名称 | |
| | -t＜target＞ | 新工程的编译目标 | 必备参数 |
| | -p＜path＞ | 新工程的保存路径 | 必备参数 |
| android update project | -t＜targe＞ | 设定工程的编译目标 | 必备参数 |
| | -p＜path＞ | 工程的保存路径 | 必备参数 |
| | -n＜name＞ | 工程名称 | |

使用 android. bat 建立 Android 工程，需要选择“开始”→“运行”→ CMD 命令，启动 CMD，并进入<Android SDK>/tools 目录，输入命令如下：

```
android create project -n HelloCommandline -k edu.hrbeu.HelloCommandline -a
HelloCommandline -t 2 -p e:\Android\workplace\HelloCommandline
```

新工程名称为 HelloCommandline，包名为 edu. hrbeu. HelloCommandline，Activity 名称是 HelloCommandline，编译目标的 ID 为 2，新工程的保存路径是 e:\Android\workplace\HelloCommandline。另一种命令输入方法如下：

```
android create project --name HelloCommandline --package edu.hrbeu.HelloCommandline
--activity HelloCommandline --target 2 --path e:\Android\workplace\HelloCommandline
```

android. bat 命令输入后的运行结果如图 3.10 所示，建立了用于 HelloCommandline 工程的目录和文件。使用命令行工具创建的 Android 工程可以移动到其他目录中，因为 android. bat 在工程创建过程中，在 local. properties 中保存了 Android SDK 的路径，因此工程建立后，建议不要改变 Android SDK 的保存路径。

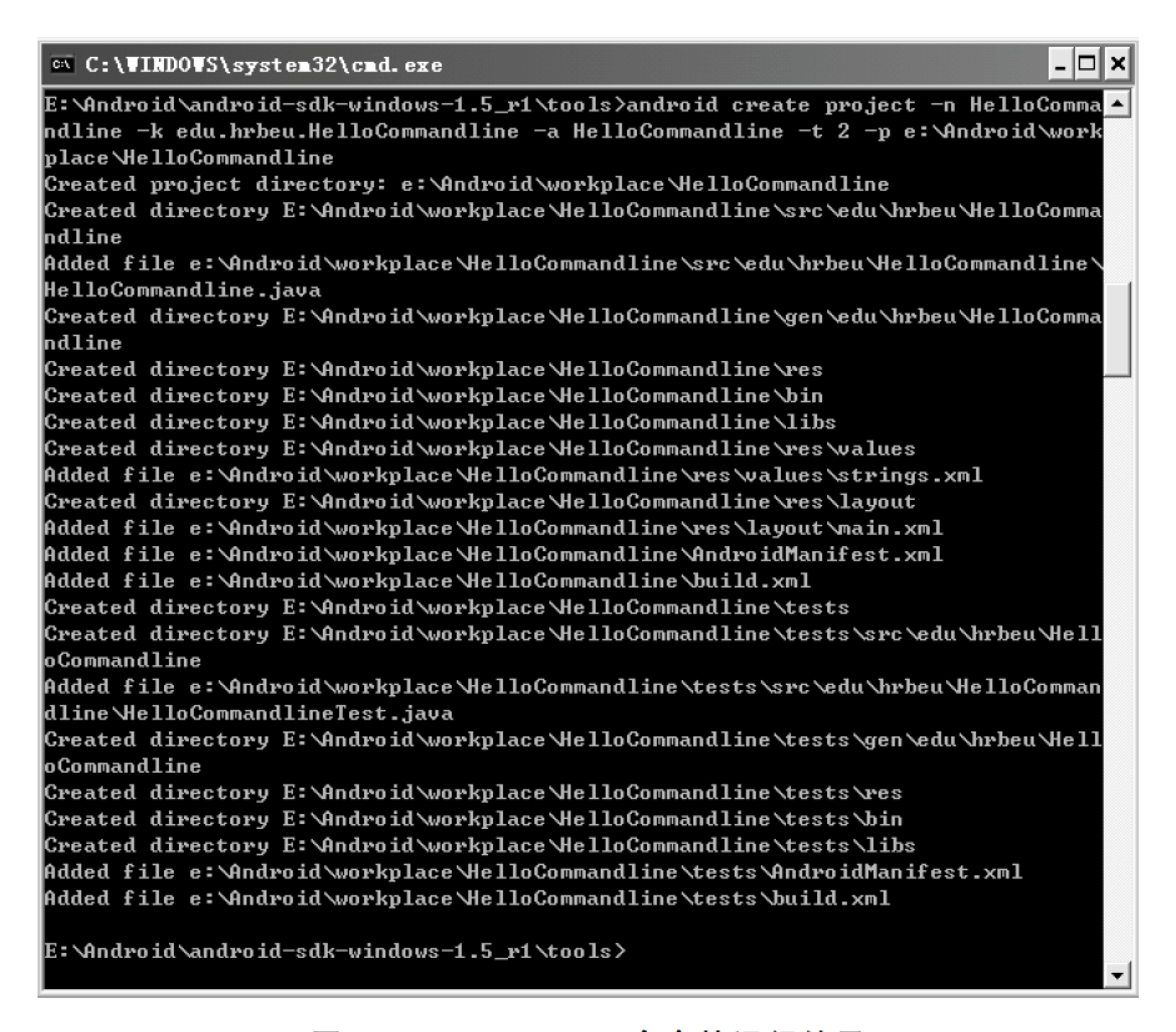



**图 3.10　android. bat 命令的运行结果**

仔细观察 android. bat 建立的目录和文件，发现其中一些在 Eclipse 开发环境中从未出现过的目录和文件，例如 build. xml、local. properties 和 tests 目录。这些新目录和文件的出现，主要是为了在构建 Android 程序时使用 Apache Ant。Apache Ant 是一个将软件编译、测试、部署等步骤联系在一起的自动化工具，多用于 Java 环境中的软件开发。在 Android 程序构建中使用 Apache Ant，可以简化程序的编译和 apk 打包过程。

表 3.2 给出了 HelloCommandline 工程的主要文件和目录，其中出现了 libs 和 tests 两个新目录，local. properties、build. properties 和 build. xml 3 个新文件。

**表 3.2　HelloCommandline 工程的文件和目录列表**

| 文　　件 | 说　　明 |
|---|---|
| AndroidManifest. xml | 应用程序声明文件 |
| build. xml | Ant 的构建文件 |
| default. properties | 保存编译目标，由 Android 工具自动建立，不可手工修改 |
| build. properties | 保存自定义的编译属性 |
| local. properties | 保存 Android SDK 的路径，仅供 Ant 使用 |
| edu\hrbeu\HelloCommandline/ HelloCommandline. java | Activity 文件 |
| bin/ | 编译脚本输出目录 |
| gen/ | 保存 Ant 自动生成文件的目录，例如 R. java |
| libs/ | 私有函数库目录 |
| res/ | 资源目录 |
| src/ | 源代码目录 |
| tests/ | 测试目录 |

libs 目录用来保存私有的函数库文件，在工程创建初期是空文件夹。

tests 目录用于测试用途，在工程创建初期，文件夹的内容是 HelloCommandline 工程所有文件和目录的一个完成拷贝。

local. properties 文件是保存 Android SDK 的路径的文件，由 Android 工具自动建立，不允许进行手工修改。该文件的主要用途是供 Apache Ant 寻找 Android SDK 的保存路径。local. properties 文件代码的第 10 行说明了 Android SDK 的路径是 E:\\Android\\android-sdk-windows-1. 5_r1。

local. properties 文件的代码如下：

```
1    # This file is automatically generated by Android Tools.
2    # Do not modify this file-- YOUR CHANGES WILL BE ERASED!
3    #
4    # This file must * NOT * be checked in Version Control Systems,
5    # as it contains information specific to your local configuration.
6
7    # location of the SDK. This is only used by Ant
8    # For customization when using a Version Control System, please read the
9    # header note.
10   sdk-location=E:\\Android\\android-sdk-windows-1.5_r1
```

说明：“\\”前一个为转义定义符，后一个为斜杠。

build.properties是保存自定义的编译属性的文件，能够修改应用程序的包名称、源代码目录和编译脚本输出目录等Apache Ant编译属性。在工程建立初期，build.properties不包含任何有效代码，用户可以手工修改文件内容。如果需要修改应用程序的包名称，可以取消build.properties文件代码的第8行的注释符号#，并将com.example.myproject替换为正确的包名称。修改源代码目录和编译脚本输出目录的位置，分别在build.properties文件代码的第11行和第14行。

build.properties文件的代码如下：

```
1   # This file is used to override default values used by the Ant build system.
2   #
3   # This file must be checked in Version Control Systems, as it is
4   # integral to the build system of your project.
5
6   # The name of your application package as defined in the manifest.
7   # Used by the 'uninstall' rule.
8   # application-package=com.example.myproject
9
10  # The name of the source folder.
11  # source-folder=src
12
13  # The name of the output folder.
14  # out-folder=bin
```

build.xml是Apache Ant的构建文件，为编译Android程序提供基础信息。build.xml文件代码的第4行～第6行分别说明了三个属性文件的名称，第8行～第14行说明了构建过程中使用到的库文件的路径。

去除注释后的build.xml文件代码如下：

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project name="HelloCommandline" default="help">
3
4       <property file="local.properties"/>
5       <property file="build.properties"/>
6       <property file="default.properties"/>
7
8       <path id="android.antlibs">
9           <pathelement path="${sdk-location}/tools/lib/anttasks.jar"/>
10          <pathelement path="${sdk-location}/tools/lib/sdklib.jar"/>
11          <pathelement path="${sdk-location}/tools/lib/androidprefs.jar"/>
12          <pathelement path="${sdk-location}/tools/lib/apkbuilder.jar"/>
13          <pathelement path="${sdk-location}/tools/lib/jarutils.jar"/>
14      </path>
15
16      <taskdef name="setup"
17          classname="com.android.ant.SetupTask"
```

```
            classpathref="android.antlibs"/>

        <setup />
    </project>
```

第二步,使用 Apache Ant 对 HelloCommondline 工程进行编译和 apk 打包。

如果读者还没有安装 Apache Ant,可以在 http://ant.apache.org/bindownload.cgi 下载,网站提供 zip、tar.gz 和 tar.bz2 三种格式下载,Windows 系统用户推荐下载 zip 格式的二进制包。笔者下载的 Apache Ant 压缩包为 apache-ant-1.7.1-bin.zip,版本号为 1.7.1,并将其解压缩在 E:\Android 目录下。

为了能 Apache Ant 正常运行,需要在 Windows 系统中添加新的环境变量,变量名和变量值参考表 3.3。Windows 系统环境变量的修改位置可选择“我的电脑”→“属性”→“高级”→“环境变量”→“系统变量”命令。

**表 3.3 新增的系统环境变量**

| 变量名 | 变量值 | 备注 |
|---|---|---|
| JAVA_HOME | C:\Program Files\Java\jdk1.6.0_12 | 新增变量 |
| ANT_HOME | E:\Android\apache-ant-1.7.1 | 新增变量 |
| ANDROID_HOME | E:\Android\android-sdk-windows-1.5_r1 | 新增变量 |
| CLASSPATH | $JAVA_HOME/jre/lib; $JAVA_HOME/lib; $JAVA_HOME/lib/tools.jar | 新增变量 |
| Path | %ANT_HOME%\bin;%JAVA_HOME%/bin; %ANDROID_HOME %\tools; | 已有变量 |

其中,JAVA_HOME 是 JDK 的安装目录,根据 JDK 实际安装位置进行修改;ANT_HOME 是 Apache Ant 的安装目录,根据 Apache Ant 实际安装位置进行修改;ANDROID_HOME 是 Android SDK 的安装目录,根据实际安装位置进行修改;CLASSPATH 是需要使用的库文件的位置;Path 是可执行文件的搜索路径,将<Apache Ant>/bin、<JDK>/bin 和<Android SDK>/tools 三个目录追加到原有的 Path 变量值中,目录之间使用分号分隔。

环境变量设置完毕后,可以在 CMD 中运行输入 Ant 命令,通过命令的输出信息判断环境变量是否设置正确。如果输出的提示包含“Unable to locate tools.jar. Expected to find it in…”,则表明没有正确设置环境变量。如果环境变量设置正确,Ant 命令的输出结果应如图 3.11 所示。

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ant
Buildfile: build.xml does not exist!
Build failed
C:\Documents and Settings\Administrator>_
```

**图 3.11 ant 命令的输出结果**

Apache Ant 已经可以正常运行了，但在构建 Android 程序前，需要介绍 Android 程序的数字签名机制。在 Android 平台上开发的所有应用程序，在安装到模拟器或手机前都必须进行数字签名。如果强行将没有数字签名的 Android 程序安装到模拟器中，将返回错误提示"Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATERS]"。

在 Eclipse 开发环境中，ADT 在将 Android 程序安装到模拟器前，已经利用内置的 debug key 为 apk 文件自动做了数字签名，这使用户无需自己生产数字签名的私钥，而能够利用 debug key 快速完成程序调试。但有一点需要注意，如果用户希望正式发布自己的应用程序，则不能使用 debug key，必须使用私有密钥对 Android 程序进行数字签名。

Apache Ant 构建 Android 应用程序支持 Debug 模式和 Release 模式两种构建模式。Debug 模式是供调试使用的构建模式，用于快速测试开发的应用程序，Debug 模式自动使用 debug key 完成数字签名；Release 模式是正式发布应用程序时使用的构建模式，生成没有数字签名的 apk 文件。

使用 Apache Ant 的 Debug 模式对 HelloCommandline 工程进行编译，生成具有 debug key 的 apk 打包文件。使用 CMD，在工程的根目录下，输入 ant debug，显示输出结果如图 3.12 所示。

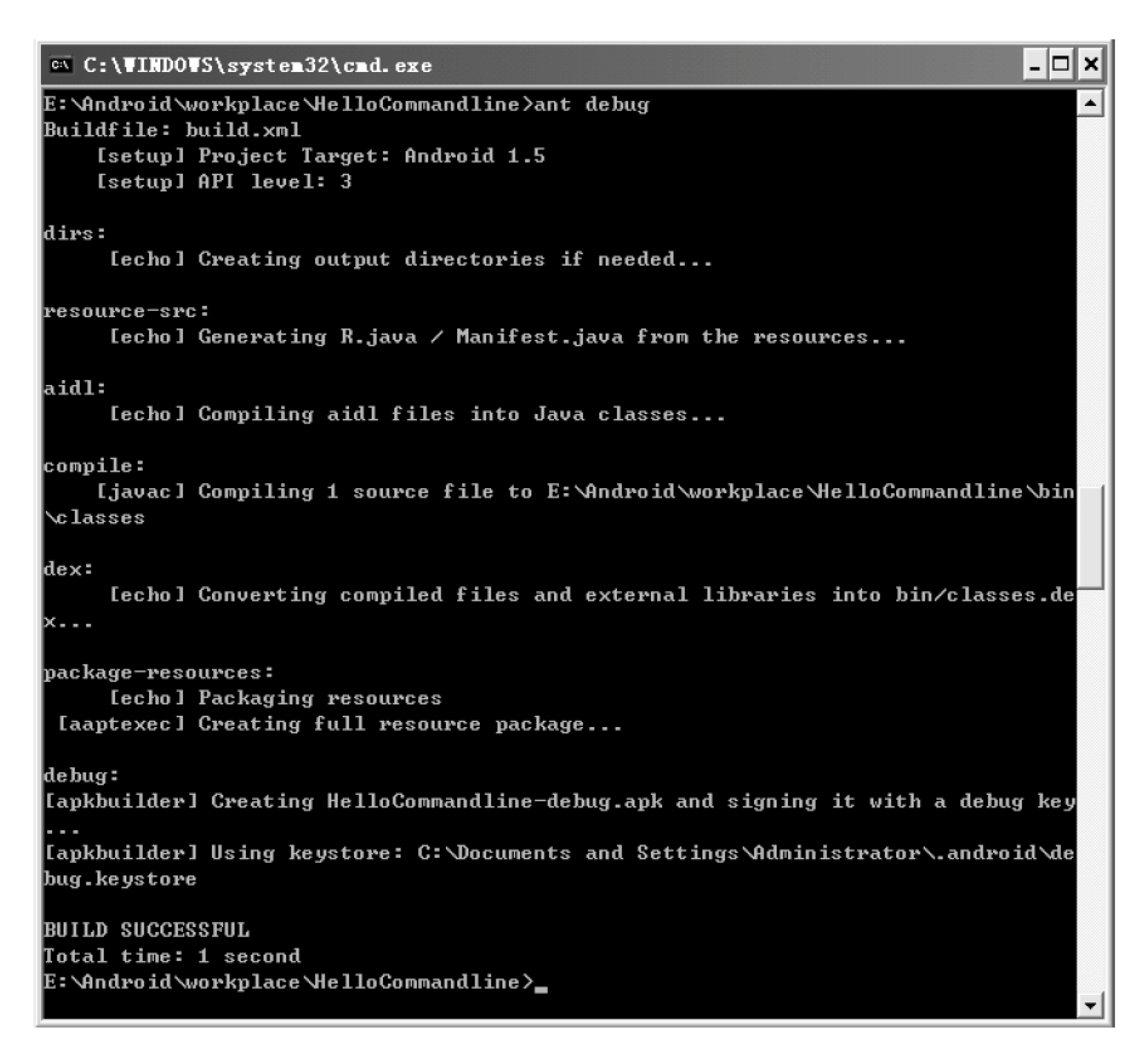



**图 3.12 ant debug 的输出结果**

命令运行后，Apache Ant 在 bin 目录中生成打包文件 HelloCommandline-debugapk。如果需要使用 Release 模式，则需在 CMD 中输入 ant release，运行后会在 bin 目录中生成打包文件 HelloCommandline-unsignedapk。

apk 文件是 Android 系统的安装程序，上传到 Android 模拟器或 Android 手机后可

以进行安装。apk 文件本身是一个 zip 压缩文件，能够使用 WinRAR、UnZip 等软件直接打开，图 3.13 是用 WinRAR 打开的 HelloCommandline-debugapk 文件。其中，res\目录用来存放资源文件；AndroidManifest.xml 是 Android 声明文件；classes.dex 是 Dalvik 虚拟机的可执行程序；resources.arsc 是编译后的二进制资源文件。

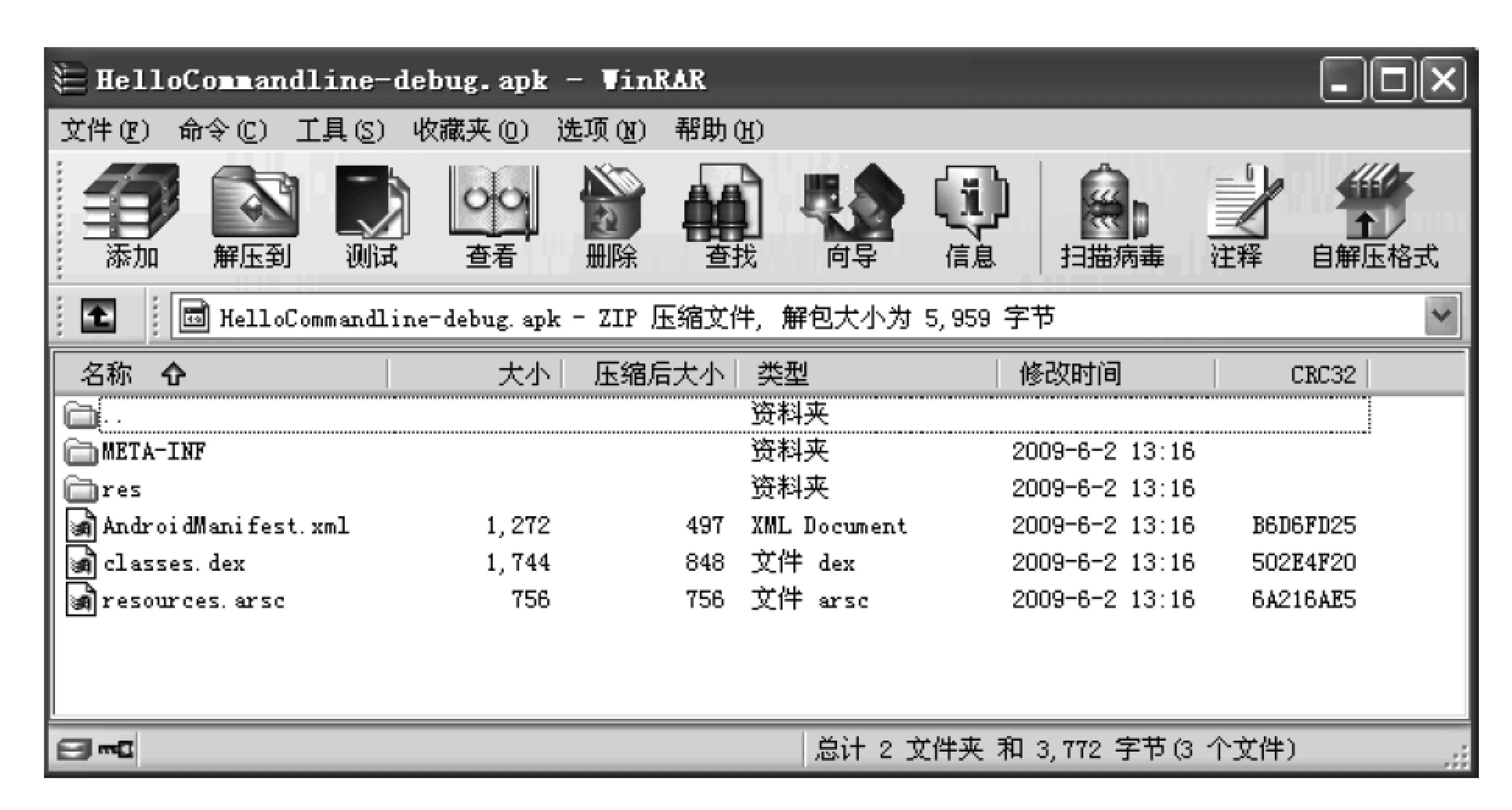



**图 3.13 HelloCommandline-debugapk 文件**

第三步，使用 adb.exe 将 HelloCommondline 工程上传到 Android 模拟器中。

从 Android SDK 1.5 版本后，用户不能通过双击 emulator.exe 启动 Android 模拟器，必须在启动模拟器时指定所使用的 AVD。AVD 的建立方法可参考 3.1 节，如果已经建立过 AVD，可以使用 android list avds 命令进行查询，查询结果如图 3.14 所示。

```
C:\WINDOWS\system32\cmd.exe

E:\Android\workplace\HelloCommandline>android list avds
Available Android Virtual Devices:
    Name: Android1.1
    Path: C:\Documents and Settings\Administrator\.android\avd\Android1.1.avd
  Target: Android 1.1 (API level 2)
    Skin: HVGA
---------
    Name: Android1.5
    Path: C:\Documents and Settings\Administrator\.android\avd\Android1.5.avd
  Target: Android 1.5 (API level 3)
    Skin: HVGA

E:\Android\workplace\HelloCommandline>_
```

**图 3.14 已有 AVD 表的查询结果**

笔者建立了两个 AVD，一个是 3.1 节建立的 Android1.5，另一个是用于区别的 Android1.1，分别对应 1.5 版本和 1.1 版本的 Android 系统。这里使用 Android1.5 的 AVD 启动模拟器，在 CMD 中输入命令 emulator -avd Android1.5，随即可以见到 Android 模拟器的启动界面。

Android 模拟器正常启动后，利用 adb.exe 工具能够把 HelloCommandline-debugapk 文件上传的模拟器中，adb.exe 工具的命令和参数可以参考附录 C。这个工具除了能够在 Android 模拟器中上传和下载文件，还能够管理模拟器状态，是调试程序时不可缺少的工具之一。在 CMD 中，进入<HelloCommandline>/bin 目录，输入命令 adb

install HelloCommandline-debugapk，完成 apk 程序上传到模拟器的过程。如果上传成功，将显示图 3.15 所示的结果。

```
C:\WINDOWS\system32\cmd.exe
E:\Android\workplace\HelloCommandline\bin>adb install HelloCommandline-debug.apk
137 KB/s (4396 bytes in 0.031s)
        pkg: /data/local/tmp/HelloCommandline-debug.apk
Success
E:\Android\workplace\HelloCommandline\bin>
```

图 3.15　向模拟器中上传 apk 文件

apk 文件上传后，并不会在 Android 模拟器上直接运行，需要用户手工启动 HelloCommandline 程序。在图 3.16 的左下角，能找到刚刚安装的 HelloCommandline 程序，单击图标即可启动应用程序。如果在模拟器中找不到 HelloCommandline 程序，尝试重新启动 Android 模拟器。因为 Android 的包管理器(Package Manager)经常仅在模拟器启动的时候检查应用程序的 AndroidManifest.xml 文件，这就导致部分上传的 Android 应用程序不能立即启动。




图 3.16　模拟器中的应用程序

用户在修改 HelloCommandline 工程代码后，需要使用 Apache Ant 重新编译和打包应用程序，并将新生成的 apk 文件上传到 Android 模拟器中。但如果新程序的包名称没有改变，则在使用 adb.exe 上传 apk 文件到模拟器时，会出现“INSTALL_FAILD_ALREADY_EXISTS”的错误提示，如图 3.17 所示。




图 3.17　安装失败的错误提示

此时，需要在模拟器中先删除原有 apk 文件，再使用 adb.exe 工具上传新的 apk 文件。有两种方法能够删除已经安装在 Android 系统中的 apk 文件，一种是使用 adb uninstall <包名称>的方法，例如删除 HelloCommandline 工程的 apk 文件，则可在 CMD 中输入命令 adb uninstall edu.hrbeu.HelloCommandline，提示 Success 则表示成功删除。另一种是使用 adb shell rm /data/app/<包名称>apk 的方法，同样以删除 HelloCommandline 工程的 apk 文件为例，在 CMD 中输入下面的命令，没有任何提示则表示删除成功。

```
adb shell rm /data/app/edu.hrbeu.HelloCommandlineapk
```

如果仅有一个 Android 模拟器在运行,用户可以一条命令完成 Android 工程编译、apk 打包和上传过程。首先启动 CMD,进入 HelloCommandline 工程的根目录下,输入 adb install,adb. exe 将自动构建工程,并使用 debug key 对工程进行签名,之后将 apk 文件上传到 Android 模拟器中。如果同时有两个或两个以上的 Android 模拟器存在,这种方法将会失败,因为 adb. exe 不能确定应该将 apk 文件上传到哪一个 Android 模拟器中。多次使用这种方法时,同样需要先删除模拟器中已有的 apk 文件。

## 习　　题

1. 简述 R. java 和 AndroidManefiest. xml 文件的用途。

2. 尝试建立一个 Android 1. 1 版本的 AVD,AVD 的名称为 MyAVD_1. 1。

3. 使用 Eclipse 建立名为 MyAndroid 的工程,包名称为 edu. hrbeu. MyAndroid,使用第 2 题中建立的 AVD,程序运行时显示 Hello MyAndroid。

4. 尝试使用命令行方式建立一个 Android 应用程序,并完成 apk 打包和程序安装过程。

# 第 4 章 chapter 4

# Android 生命周期

Android 生命周期是从程序启动到程序终止的全过程。通过本章的学习可以让读者深入理解 Android 系统管理生命周期的必要性，并以 Activity 为例说明 Android 系统如何管理程序组件的生命周期。对调试方法和工具的介绍，有助于程序开发人员快速找到程序中的错误，而且可以对特殊的事件回调函数进行调试。

**本章学习目标：**

- 了解 Android 系统的进程优先级的变化方式
- 了解 Android 系统的四大基本组件
- 了解 Activity 的生命周期中各状态的变化关系
- 掌握 Activity 事件回调函数的作用和调用顺序
- 掌握 Android 应用程序的调试方法和工具

## 4.1 程序生命周期

软件生命周期是软件从产生到废弃所历经的几个阶段，一般包括可行性分析、开发计划、需求分析与设计、编码、测试和维护等过程。Android 的程序生命周期与软件生命周期的定义不同，它指的是 Android 进程从启动到终止的所有阶段，也就是 Android 程序启动到停止的全过程。

Android 系统一般是运行在资源受限的硬件平台上，因此资源管理对 Android 系统至关重要。Android 系统主动管理资源，为了保证高优先级程序正常运行，可以在无任何警告的情况下终止优先级低的程序，并回收其使用的系统资源。因此 Android 程序并不能控制自身的生命周期，而完全是由 Android 系统进行调度和控制的。

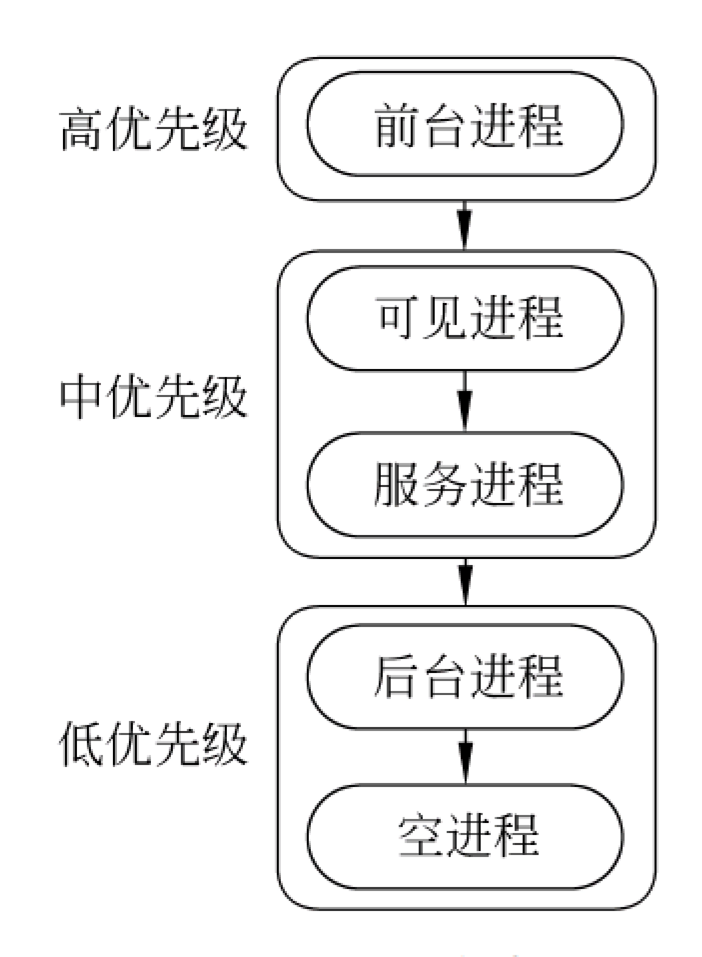



**图 4.1　进程优先级**

Android 系统尽可能地不主动终止应用程序，即使生命周期结束的程序也会保存在内存中，以便再次快速启动。但在内存紧张时，系统会根据进程的优先级清除进程，回收系统资源。Android 系统中的进程优先级如图 4.1 所示，优先级由高到低分别为前台进程、可见进程、服务进程、后台进程和空进程。

### 1. 前台进程

前台进程是 Android 系统中最重要的进程，是与用户正在交互的进程，包含以下 4 种情况：

- 进程中的 Activity 正在与用户进行交互；
- 进程服务被 Activity 调用，而且这个 Activity 正在与用户进行交互；
- 进程服务正在执行声明周期中的回调函数，如 onCreate()、onStart() 或 onDestroy()；
- 进程的 BroadcastReceiver 正在执行 onReceive()函数。

Android 系统在多个前台进程同时运行时，可能会出现资源不足的情况，此时会清除部分前台进程，保证主要的用户界面能够及时响应。

### 2. 可见进程

可见进程指部分程序界面能够被用户看见，却不在前台与用户交互，不响应界面事件的进程。例如，新启动的 Android 程序将原有程序部分遮挡，原有程序从前台进程变为可见进程。另外，如果一个进程包含服务，且这个服务正在被用户可见的 Activity 调用，此进程同样被视为可见进程。一般 Android 系统会存在少量的可视进程，只有在极端的情况下，Android 系统才会为保证前台进程的资源而清除可视进程。

### 3. 服务进程

一个包含已启动服务的进程就是服务进程。服务没有用户界面，不与用户直接交互，但能够在后台长期运行，提供用户所关心的重要功能，例如播放 MP3 文件或从网络下载数据。因此，Android 系统除非不能保证前台进程或可视进程所必要的资源，否则不强行清除服务进程。

### 4. 后台进程

如果一个进程不包含任何已经启动的服务，而且没有任何用户可见的 Activity，则这个进程就是后台进程。例如，在一个仅有 Activity 组件的进程中，用户启动了新应用程序，使完全遮挡住了当前进程的 Activity，则这个进程便成为了后台进程。一般情况下，Android 系统中存在数量较多的后台进程，在系统资源紧张时，系统将优先清除用户较长时间没有见到的后台进程。

### 5. 空进程

空进程是不包含任何活跃组件的进程，例如一个仅有 Activity 组件的进程，当用户关闭 Activity 后，这个进程就成为空进程。空进程在系统资源紧张时会被首先清除，但为了提高 Android 系统应用程序的启动速度，Android 系统会将空进程保存在系统内存中，在用户重新启动该程序时，空进程会被重新使用。

在 Android 中，进程的优先级取决于所有组件中的优先级最高的部分。例如，在进

程中同时包含部分可见的 Activity 和已经启动的服务，则该进程是可见进程，而不是服务进程。另外，进程的优先级会根据与其他进程的依赖关系而变化。例如，进程 A 的服务被进程 B 调用，如果调用前进程 A 是服务进程，进程 B 是前台进程，则调用后进程 A 也具有前台进程的优先级。

## 4.2 Android 组件

Android 应用程序由组件组成，组件是可以被调用的基本功能模块。Android 系统利用组件实现程序内部或程序间的模块调用，以解决代码复用的问题，这是 Android 系统非常重要的特性。在程序设计时，在 AndroidManifest. xml 中声明可共享的组件，声明后其他应用程序可以直接调用这些共享组件。例如程序 A 实现了文件压缩的功能，并对外共享了这个组件，程序 B 则不必再开发文件压缩功能，而直接调用程序 A 的共享组件即可。

但这种特性存在一个问题，如果共享组件所在的进程没有启动，这个共享组件如何被其他程序调用？为了解决这一问题，Android 系统必须能够在其他程序调用共享组件时，能够直接启动被调用的共享组件。因此，Android 系统没有使用常见的应用程序入口点（类似于 Java 程序的 Main 函数）的方法，而是允许共享组件被 Android 系统直接实例化，从而保证能够调用进程没有启动的共享组件。

Android 系统有 4 个重要的组件，分别是 Activity、Service、BroadcaseReceiver 和 ContentProvider。

Activity 是 Android 程序的呈现层，显示可视化的用户界面，并接收与用户交互所产生的界面事件，与“窗体”的概念非常相似。Android 应用程序可以包含一个或多个 Activity，一般在程序启动后会呈现一个 Activity，用于提示用户程序已经正常启动。Activity 在界面上的表现形式一般是全屏窗体，也可以是非全屏悬浮窗体或对话框。

Service 一般用于没有用户界面，但需要长时间在后台运行的应用。例如在播放 MP3 音乐时，使用 Service 播放 MP3 音乐，可以在关闭播放器界面的情况下长时间播放 MP3 音乐，并通过对外公开 Service 的通信接口，控制 MP3 音乐播放的启动、暂停和停止。

BroadcaseReceiver 是用来接受并响应广播消息的组件。大部分广播消息是由系统产生的，例如时区改变、电池电量低或语言选项改变等，但应用程序也可以产生广播消息，例如数据下载完毕等。Broadcast receiver 不包含任何用户界面，但可以通过启动 Activity 或者 Notification 通知用户接收到重要信息。Notification 能够通过多种方法提示用户，包括闪动背景灯、震动设备、发出声音或在状态栏上放置一个持久的图标等。

ContentProvider 是 Android 系统提供的一种标准的共享数据的机制，应用程序可以通过 ContentProvider 访问其他应用程序的私有数据。私有数据可以是存储在文件系统中的文件，也可以是 SQLite 中的数据库。Android 系统内部也提供一些内置的 ContentProvider，能够为应用程序提供重要的数据信息，例如联系人信息和通话记录等。

Android 系统通过组件机制，有效地降低了应用程序的耦合性，使向其他应用程序共享私有数据（ContentProvider）和调用其他程序的私有模块（Service）成为可能。

所有 Android 组件都具有自己的生命周期，称为组件声明周期，是从组件建立到组

件销毁的整个过程。在这个过程中,组件会在可见、不可见、活动、非活动等状态中不断变化,4.3 节将主要对 Activity 的生命周期进行详细介绍。

## 4.3 Activity 生命周期

Activity 生命周期指 Activity 从启动到销毁的过程,在这个过程中,Activity 一般表现为 4 种状态,分别是活动状态、暂停状态、停止状态和非活动状态。(1)活动状态,当 Activity 在用户界面中处于最上层,用户完全看不到,能够与用户进行交互,则这个 Activity 处于活动状态。(2)暂停状态,当 Activity 在界面上被部分遮挡,该 Activity 不再处于用户界面的最上层,且不能够与用户进行交互,则这个 Activity 处于暂停状态。(3)停止状态,当 Activity 在界面上能完全不被用户看到,也就是说这个 Activity 被其他 Activity 全部遮挡,则这个 Activity 处于停止状态。(4)非活动状态,活动状态、暂停状态和停止状态是 Activity 的主要状态,不在以上三种状态中的 Activity 则处于非活动状态。

Activity 的 4 种状态的变换关系如图 4.2 所示。Activity 启动后处于活动状态,此时的 Activity 位于界面的最上层,是与用户正在进行交互的组件,因此 Android 系统会努力保证处于活动状态 Activity 的资源需求,资源紧张时可终止其他状态的 Activity;如果用户启动了新的 Activity,部分遮挡了当前的 Activity,或新的 Activity 是半透明的,则当前的 Activity 转换为暂停状态,Android 系统仅在为处于活动状态的 Activity 释放资源时才终止处于暂停状态的 Activity;如果用户启动新的 Activity 完全遮挡了当前的 Activity,则当前的 Activity 转变为停止状态,停止状态的 Activity 将优先被终止;活动状态的 Activity 被用户关闭后,或暂停状态或停止状态的 Activity 被系统终止后,Activity 便进入了非活动状态。

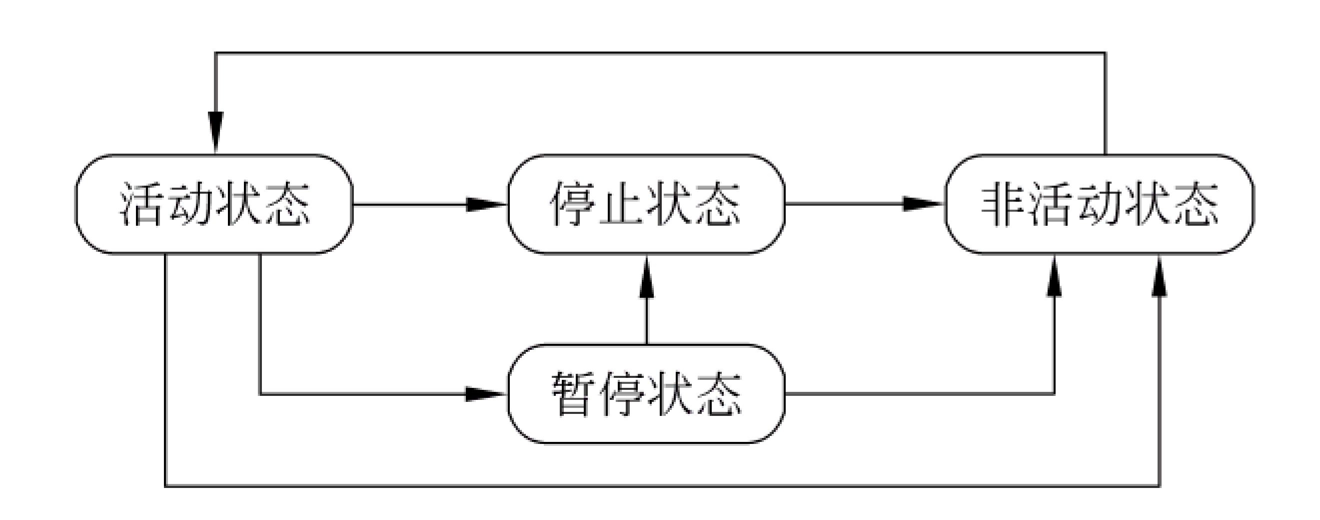



**图 4.2 Activity 状态变换图**

为能够更好地理解 Activity 的生命周期,还需要对 Activity 栈做一个简要的介绍。Activity 栈保存了已经启动且没有终止的所有 Activity,并遵循"后进先出"的规则。如图 4.3 所示,栈顶的 Activity 处于活动状态,除栈顶以外的其他 Activity 处于暂停状态或停止状态,而被终止的 Activity 或已经出栈的 Activity 的 Activity 则不在栈内。

Activity 的状态与其在 Activity 栈的位置有着密切的关系,不仅如此,Android 系统在资源不足时,也是通过 Activity 栈来选择哪些 Activity 是可以被终止的。一般来讲,Android 系统会优先选终止处于停止状态,且位置靠近栈底的 Activity,因为这些 Activity 启动顺序最靠前,而且在界面上用户是看不到的。

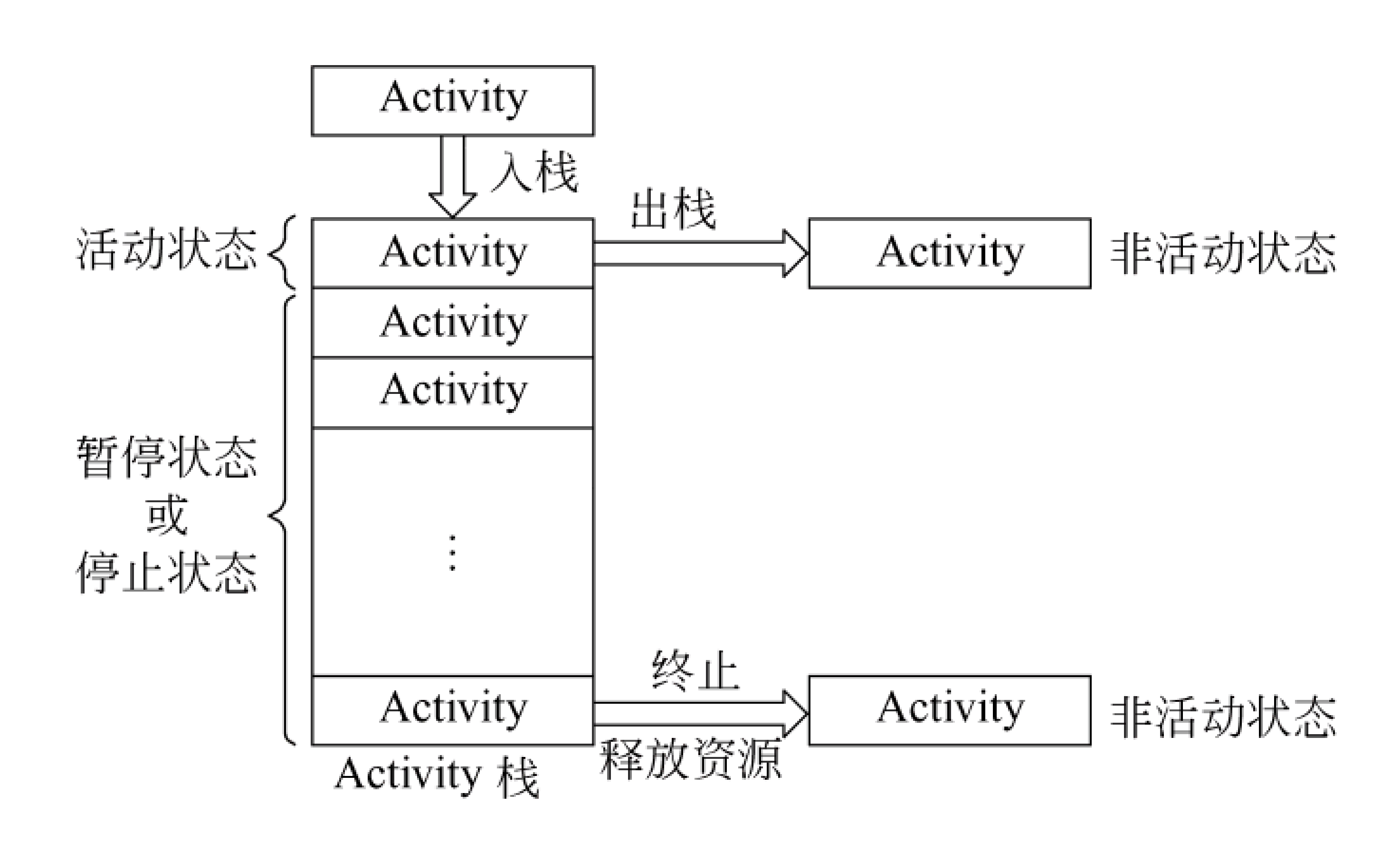



**图 4.3　Activity 栈**

随着用户在界面的操作和 Android 系统对资源的管理，Activity 不断变化在 Activity 栈的位置，其状态也不断在 4 种状态中转变。为了能够让 Android 程序了解自身状态的变化，Android 系统提供了多个的事件回调函数，在事件回调函数添加相关代码，就可以在 Activity 状态变化时完成适当的工作。下面的代码给出了 Activity 的主要事件回调函数。

```
public class MyActivity extends Activity {
      protected void onCreate(Bundle savedInstanceState);
      protected void onStart();
      protected void onRestart();
      protected void onResume();
      protected void onPause();
      protected void onStop();
      protected void onDestroy();
 }
```

这些事件回调函数何时被调用，具体用途是什么，以及是否可以被 Android 系统终止，可以参考表 4.1。

**表 4.1　Activity 生命周期的事件回调函数**

| 函数 | 是否可终止 | 说　　明 |
|---|---|---|
| onCreate() | 否 | Activity 启动后第一个被调用的函数，常用来进行 Activity 的初始化，例如创建 View、绑定数据或恢复信息等 |
| onStart() | 否 | 当 Activity 显示在屏幕上时，该函数被调用 |
| onRestart() | 否 | 当 Activity 从停止状态进入活动状态前，调用该函数 |
| onResume() | 否 | 当 Activity 能够与用户交互，接受用户输入时，该函数被调用。此时的 Activity 位于 Activity 栈的栈顶 |
| onPause() | 是 | 当 Activity 进入暂停状态时，该函数被调用。一般用来保存持久的数据或释放占用的资源 |
| onStop() | 是 | 当 Activity 进入停止状态时，该函数被调用 |
| onDestroy() | 是 | 在 Activity 被终止前，即进入非活动状态前，该函数被调用 |

除了 Activity 生命周期的事件回调函数以外，还有 onRestoreInstanceState() 和 onSaveInstanceState()两个函数经常会被使用，用于保存和恢复 Activity 的状态信息，例如用户在界面中选择的内容或输入的数据等。这两个函数不是生命周期的事件回调函数，不会因为 Activity 的状态变化而被调用，但在下述的情况下会被调用：Android 系统因为资源紧张需要终止某个 Activity，但这个 Activity 在未来的某一时刻还会显示在屏幕上。

举个例子说明这两个函数是如何被调用的。如用户启动 Activity A，然后直接又启动 Activity B，这时系统资源不足，需要终止 Activity A，则会调用 Activity A 的 onSaveInstanceState()来保存 Activity A 的状态信息；当用户关闭了 Activity B 时，Activity B 的 onSaveInstanceState()不会被调用，因为是用户主动关闭 Activity B 而不是系统终止，而且 Activity B 不会再显示在屏幕上；当 Activity A 重新显示在屏幕上后，Activity A 会调用 onRestoreInstanceState()用以恢复之前保存的 Activity A 的状态信息。

Activity 状态保存和恢复函数 onRestoreInstanceState()和 onSaveInstanceState()的说明参见表 4.2。

**表 4.2 Activity 状态保存/恢复的事件回调函数**

| 函数 | 是否可终止 | 说明 |
| --- | --- | --- |
| onSaveInstanceState() | 否 | Android 系统因资源不足终止 Activity 前调用该函数，用以保存 Activity 的状态信息，供 onRestoreInstanceState()或 onCreate()恢复之用 |
| onRestoreInstanceState() | 否 | 恢复 onSaveInstanceState()保存的 Activity 状态信息，在 onStart()和 onResume ()之间被调用 |

函数 onSaveInstanceState()保存的 Activity 状态信息，也可以在 onCreate()函数中进行恢复，但 onRestoreInstanceState()使用起来更加方便，也使得代码更加整洁。因为使用者可以在 onCreate()函数中完成初始化工作结束后，再在 onRestoreInstanceState()决定是否使用默认设置或恢复保存的数据。

在 Activity 的生命周期中，并不是所有的事件回调函数都将被执行，但如果被调用则一定遵循图 4.4 所描述的调用顺序。

从图 4.4 中可知，Activity 的生命周期可分为全生命周期、可视生命周期和活动生命周期。每种生命周期中包含不同的事件回调函数。

全生命周期是从 Activity 建立到销毁的全部过程，始于 onCreate()，结束于 onDestroy()。一般情况下，使用者在 onCreate()中初始化 Activity 所能使用的全局资源和状态，并在 onDestroy()中释放这些资源。例如，Activity 中使用后台线程下载网络数据，则在 onCreate()中创建线程，在 onDestroy()中停止并销毁线程。在一些极端的情况下，Android 系统会不调用 onDestroy()函数，而直接终止进程。

可视生命周期是 Activity 在界面上从可见到不可见的过程，开始于 onStart()，结束于 onStop()。onStart()一般用来初始化或启动与更新界面相关的资源。onStop()一般

用来暂停或停止一切与更新用户界面相关的线程、计时器和服务，因为在调用 onStop()后，Activity 对用户不再可见，更新用户界面也就没有任何实际意义。onRestart()函数在 onSart()前被调用，用来在 Activity 从不可见变为可见的过程中，进行一些特定的处理过程。因为 Activity 不断从可见变为不可见，再从不可见变为可见，所以 onStart()和 onStop()会被多次调用。另外，onStart()和 onStop()也经常被用来注册和注销 BroadcastReceiver，例如使用者可以在 onStart()中注册一个 BroadcastReceiver，用来监视某些重要的广播消息，并使用这些消息更新用户界面中的相关内容，并可以在 onStop()中注销 BroadcastReceiver。

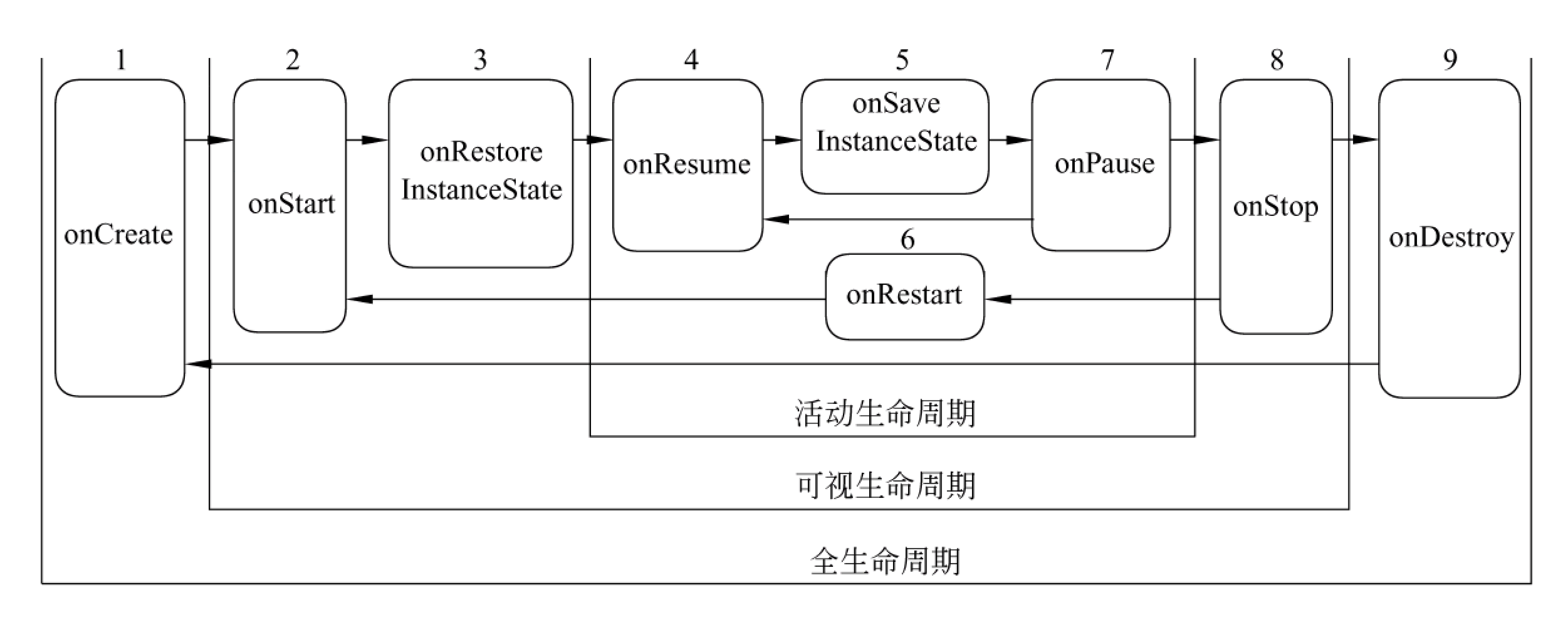



**图 4.4 Activity 事件回调函数的调用顺序**

活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段，开始于 onResume()，结束于 onPause()。例如，在手机进入休眠状态时，处于活动生命周期的 Activity 会调用 onPause()，当手机从休眠状态被唤醒后，则会调用 onResume()。因为在 Activity 的状态变换过程中 onResume()和 onPause()经常被调用，因此这两个函数中应使用更为简单、高效的代码。

在表 4.1 和表 4.2 中，“是否可终止”表示事件回调函数在执行过程中或返回后是否可以被 Android 系统终止，否定答案表示在函数从被调用后，直到函数返回前，Android 系统不能够终止该进程；肯定答案表示当前的函数返回后，Android 系统可以随时终止该进程。从图 4.4 的 Activity 事件回调函数的调用顺序上分析，onPause()是第一个被标识为“可终止”的函数，因此在 onPause()返回后，后续的 onStop()和 onDestroy()随时能被 Android 系统终止，也就是说系统需要终止进程时，onPause()是唯一一个能够完整运行事件回调函数。因此，onPause()常用来保存持久数据，如界面上的用户的输入信息等。很多时候使用者不能清楚何时该使用 onPause()，何时该使用 onSaveInstanceState()，因为两个函数都可以用来保存界面的用户输入数据。其主要区别在于两个函数保存数据的性质和方法不同，onPause()一般用于保存持久性数据，并将数据保存在存储设备上的文件系统或数据库系统中的；而 onSaveInstanceState()主要用来保存动态的状态信息，信息一般保存在 Bundle 中。Bundle 是能够保存多种格式数据的对象，在 onSaveInstanceState()保存在 Bundle 中的数据，系统在调用 onRestoreInstanceState()和 onCreate()时，

会同样利用 Bundle 将数据传递给函数。

为了能够更好地理解 Activity 事件回调函数的调用顺序，下面建立一个新的 Android 工程来进行说明。工程名称 ActivityLifeCycle，包名称 edu. hrbeu. ActivityLifeCycle，Activity 名称 ActivityLifeCycle。下面的代码是 ActivityLifeCycle. java 文件的代码，其余代码使用自动生成的代码。

```
package edu.hrbeu.ActivityLifeCycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ActivityLifeCycle extends Activity {
    private static String TAG= "LIFTCYCLE";
    @Override                //完全生命周期开始时被调用,初始化 Activity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.i(TAG, "(1) onCreate()");
    }

    @Override                //可视生命周期开始时被调用,对用户界面进行必要的更改
    public void onStart() {
        super.onStart();
        Log.i(TAG, "(2) onStart()");
    }

    @Override                //在 onStart()后被调用,用于恢复 onSaveInstance State()保存的用
                               户界面信息
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        Log.i(TAG, "(3) onRestoreInstanceState()");
    }

    @Override                //在活动生命周期开始时被调用,恢复被 onPause()停止的用于
                               界面更新的资源
    public void onResume() {
        super.onResume();
        Log.i(TAG, "(4) onResume()");
    }

    @Override                //在 onResume()后被调用,保存界面信息
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
```

```
            Log.i(TAG, "(5) onSaveInstanceState()");
        }

        @Override              //在重新进入可视生命周期前被调用,载入界面所需要的更改信息
        public void onRestart() {
            super.onRestart();
            Log.i(TAG, "(6) onRestart()");
        }

        @Override      //在活动生命周期结束时被调用,用来保存持久的数据或释放占用的资源
        public void onPause() {
            super.onPause();
            Log.i(TAG, "(7) onPause()");
        }

        @Override
                  //在可视生命周期结束时被调用,一般用来保存持久的数据或释放占用的资源
        public void onStop() {
            super.onStop();
            Log.i(TAG, "(8) onStop()");
        }

        @Override              //在完全生命周期结束时被调用,释放资源,包括线程、数据连接等
        public void onDestroy() {
            super.onDestroy();
            Log.i(TAG, "(9) onDestroy()");
        }
    }
```

上面的程序主要通过在生命周期函数中添加“日志点”的方法进行调试,程序的运行结果将会显示在 LogCat 中。LogCat 和“日志点”的使用方法,请参考 4.4.1 节内容。为了显示结果易于观察和分析,在 LogCat 设置过滤器 LifeCycleFilter,过滤方法选择 by Log Tag,过滤关键字为 LIFTCYCLE。日志信息中的数字标识与图 4.4 的函数编号一致。

1) 全生命周期

为了观察 Android 程序启动和关闭时调用生命周期函数的顺序,首先正常启动 ActivityLifeCycle,然后按下模拟器的“返回键”,关闭 ActivityLifeCycle。LogCat 的输出结果如图 4.5 所示。

从图 4.5 可以得知,函数调用顺序如下:onCreate()→onStart()→onResume()→onPause()→onStop()→onDestroy()。

在 Activity 启动时,系统首先调用 onCreate()函数分配资源,然调用 onStart()将 Activity 显示在屏幕上,之后调用 onResume()获取屏幕焦点,使 Activity 能够接受用户

的输入，这时用户就能够正常使用这个 Android 程序了。用户单击“回退键”或 Activity 调用 finish()函数时，会导致 Activity 关闭，系统会相继调用 onPause()、onStop()和 onDestroy()，释放资源并销毁进程。因为 Activity 关闭后，除非用户重新启动应用程序，否则这个 Activity 不会在出现在屏幕上，因此系统直接调用 onDestroy()销毁了进程，且没有调用 onSaveInstanceState()函数来保存 Acitivity 状态。

Log (4) LifeCycleFilter

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 01:22:40.741 | I | 779 | LIFECYCLE | (1) onCreate() |
| 06-07 01:22:40.753 | I | 779 | LIFECYCLE | (2) onStart() |
| 06-07 01:22:40.760 | I | 779 | LIFECYCLE | (4) onResume() |
| 06-07 01:23:30.192 | I | 779 | LIFECYCLE | (7) onPause() |
| 06-07 01:23:30.473 | I | 779 | LIFECYCLE | (8) onStop() |
| 06-07 01:23:30.490 | I | 779 | LIFECYCLE | (9) onDestroy() |

**图 4.5　启动和关闭 ActivityLifeCycle 的 LogCat 输出**

2）可视生命周期

在 Activity 启动后，如果启动其他的程序，原有的 Activity 会被新启动程序的 Activity 完全遮挡，因此原有 Activity 会进入停止状态；如果将新启动的程序关闭，则原有 Activity 从停止状态恢复到活动状态。为了能够分析上述状态转换过程中的函数调用顺序，首先正常启动 ActivityLifeCycle，然后通过“呼出/接听键”启动内置的拨号程序，再通过“返回键”退出拨号程序，使 ActivityLifeCycle 重新显示在屏幕中。LogCat 的输出结果如图 4.6 所示，ActivityLifeCycle 启动时调用的函数没有在 LogCat 中显示。

Log (8) LifeCycleFilter

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 02:23:01.530 | I | 779 | LIFECYCLE | (5) onSaveInstanceState() |
| 06-07 02:23:01.530 | I | 779 | LIFECYCLE | (7) onPause() |
| 06-07 02:23:01.609 | I | 779 | LIFECYCLE | (8) onStop() |
| 06-07 02:23:04.483 | I | 779 | LIFECYCLE | (6) onRestart() |
| 06-07 02:23:04.499 | I | 779 | LIFECYCLE | (2) onStart() |
| 06-07 02:23:04.499 | I | 779 | LIFECYCLE | (4) onResume() |

**图 4.6　可视生命周期的 LogCat 输出**

从图 4.6 可以得知，函数调用顺序如下：onSaveInstanceState()→onPause()→onStop()→onRestart()→onStart()→onResume()。

Activity 启动后，当内置的拨号程序被启动时，原有的 Activity 被完全覆盖，系统首先调用 onSaveInstanceState()函数保存 Activity 状态，因为界面上不可见的 Activity 可能被系统终止；系统然后调用 onPause()和 onStop()，停止对不可见 Activity 的更新。在用户关闭拨号程序后，系统调用 onRestart()恢复需要界面上需要更新的信息，然后调用 onStart()和 onResume()重新显示 Activity，并接受用户交互。系统调用了 onSaveInstanceState()保存 Activity 的状态，因为进程没有被终止，所以没有调用 onRestoreInstanceState()恢复保存的 Activity 状态。

如果用户在 Dev Tools→Development Settings→Immediately destroy activities (IDA)下开启 IDA，被其他程序遮挡的 Activity 会被立即终止，这样被遮挡的 Activity 重新显示在屏幕上时，系统会调用 onRestoreInstanceState()恢复 Activity 销毁前的状态。

从图 4.7 可以得知，函数调用顺序如下：onSaveInstanceState()→onPause()→onStop()→onDestroy()→onCreate()→onStart()→onRestoreInstanceState()→onResume()。

Log (10) | LifeCycleFilter

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 04:21:30.806 | I | 779 | LIFECYCLE | (5) onSaveInstanceState() |
| 06-07 04:21:30.806 | I | 779 | LIFECYCLE | (7) onPause() |
| 06-07 04:21:31.587 | I | 779 | LIFECYCLE | (8) onStop() |
| 06-07 04:21:31.602 | I | 779 | LIFECYCLE | (9) onDestroy() |
| 06-07 04:21:34.040 | I | 779 | LIFECYCLE | (1) onCreate() |
| 06-07 04:21:34.074 | I | 779 | LIFECYCLE | (2) onStart() |
| 06-07 04:21:34.074 | I | 779 | LIFECYCLE | (3) onRestoreInstanceState() |
| 06-07 04:21:34.074 | I | 779 | LIFECYCLE | (4) onResume() |

**图 4.7 开启 IDA 的可视生命周期 LogCat 输出**

与关闭 IDA 相比，Activity 被拨号程序遮挡后，系统调用了 onDestroy()终止了 ActivityLifeCycle 进程，这个函数的调用完全是因为 Android 系统开启了 IDA 进行调试；在拨号程序被关闭后，ActivityLifeCycle 的 Activity 应该重新显示在屏幕上，但因为系统已经终止了进程，因此需要重新启动 ActivityLifeCycle，因为 Activity 被系统终止，因此系统调用了 onRestoreInstanceState()来恢复之前 onSaveInstanceState()函数保存的 Activity 状态，其他的函数调用顺序与程序启动过程的调用顺序相同。

3）活动生命周期

Activity 启动后，如果被其他程序部分遮挡，或手机进入休眠状态，这个 Activity 进入暂停状态。为了分析活动生命周期中的函数调用顺序，首先启动 ActivityLifeCycle，然后通过“挂断键”使模拟器进入休眠状态，然后再通过“挂断键”唤醒模拟器。此时 LogCat 的输出结果如图 4.8 所示。

Log (9) | LifeCycleFilter

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 04:47:13.673 | I | 779 | LIFECYCLE | (5) onSaveInstanceState() |
| 06-07 04:47:13.673 | I | 779 | LIFECYCLE | (7) onPause() |
| 06-07 04:47:15.038 | I | 779 | LIFECYCLE | (4) onResume() |

**图 4.8 活动生命周期的 LogCat 输出**

从图 4.8 可以得知，函数调用顺序如下：onSaveInstanceState()→onPause()→onResume()。

Activity 进入休眠状态前，系统首先调用 onSaveInstanceState()保存 Activity 的状态，然后调用 onPause()停止与用户交互；在系统被唤醒后，系统调用 onResume()恢复与用户的交互。

## 4.4 程序调试

在 Android 程序开发过程中，出现错误(Bug)是不可避免的事情。一般情况下，语法错误会被集成开发环境检测到，并提示使用者错误的位置以及修改方法。但逻辑错误就

不那么容易发现了，通常只有将程序在模拟器或硬件设备上运行才能够发现。逻辑错误的定位和分析是件困难的事情，尤其程序是代码量较大且结构复杂的应用程序，仅凭直觉很难快速找到并解决问题。因此，Android 系统提供了几种调试工具，用于定位、分析及修复程序中出现的错误，这些工具包括 LogCat 和 DevTools。

### 4.4.1 LogCat

LogCat 是用来获取系统日志信息的工具，并可以显示在 Eclipse 集成开发环境中。LogCat 能够捕获的信息包括 Dalvik 虚拟机产生的信息、进程信息、ActivityManager 信息、PackagerManager 信息、Homeloader 信息、WindowsManager 信息、Android 运行时信息和应用程序信息等等。在 Eclipse 的默认开发模式下没有 LogCat 的显示页，用户可以使用 Windows→Show View→Other 打开 Show View 的选择菜单，然后在 Andoird→LogCat 中选择 LogCat，如图 4.9 所示。

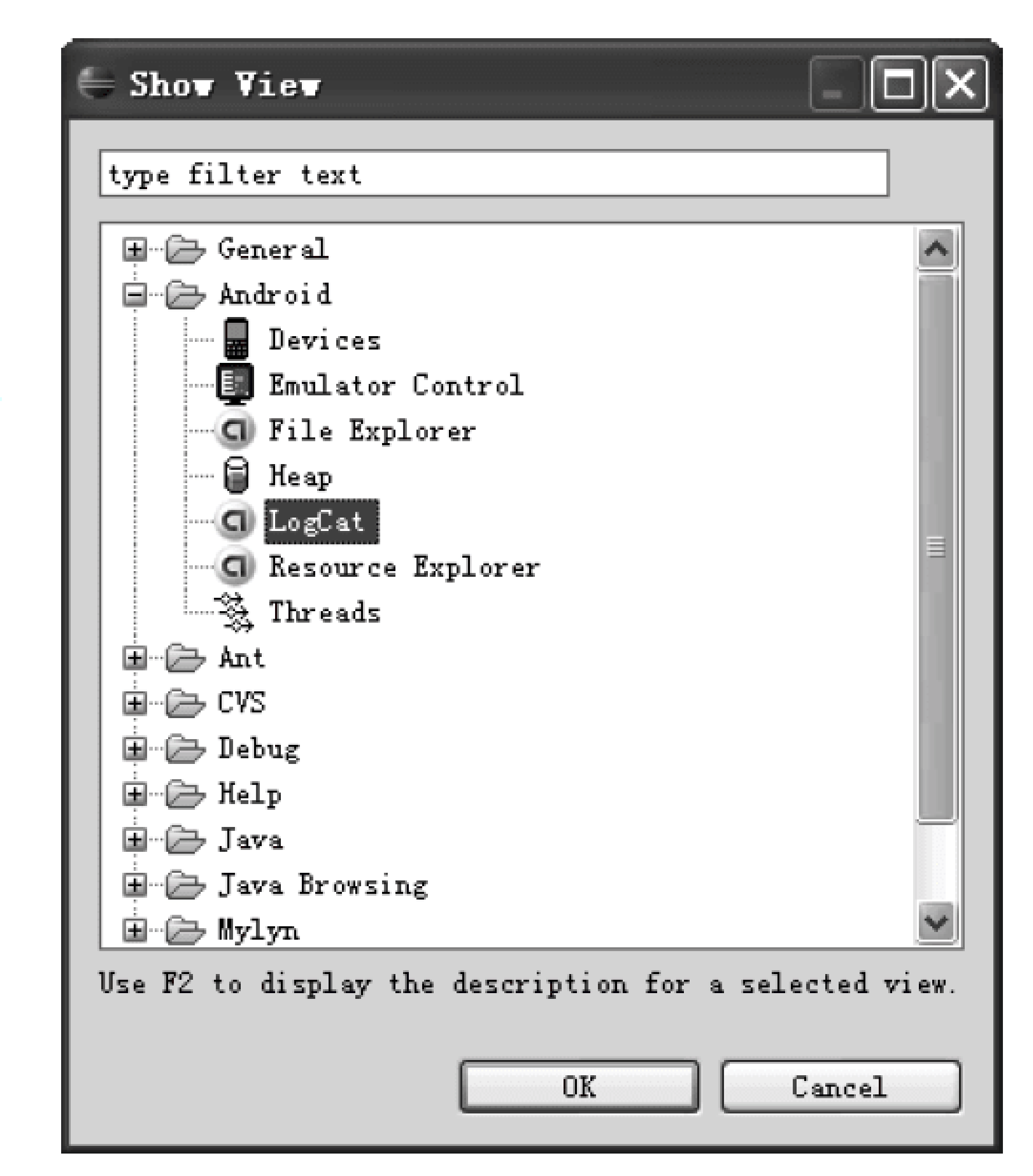



**图 4.9　Show View 中选择 LogCat**

这样，LogCat 便显示在 Eclipse 的下方区域，如图 4.10 所示。LogCat 的右上方的 5 个字母[V]、[D]、[I]、[W]和[E]，表示 5 种不同类型的日志信息，分别是详细(Verbose)信息、调试(Debug)信息、通告(Info)信息、警告(Warn)和错误(Error)信息。不同类型日志信息的级别是不相同的，级别最高的是错误信息，其次是警告信息，然后是通知信息和调试信息，级别最低的是详细信息。在 LogCat 中，用户可以通过 5 个字母图标选择显示的信息类型，级别高于所选类型的信息也会在 LogCat 中显示，但级别低于所选类型的信息则不会被显示。

即使用户指定了所显示日志信息的级别，仍然会产生很多日志信息，很容易让用户不知所措。LogCat 还提供了“过滤”功能，在右上角的“+”号和“−”号，分别是添加和删除过滤器。用户可以根据日志信息的标签(Tag)、产生日志的进程编号(Pid)或信息等级(Level)，对显示的日志内容进行过滤。

在 Android 程序调试过程中，首先需要引入 android.util.Log 包，然后使用 Log.v( )、Log.d( )、Log.i( )、Log.w( )和 Log.e( )五个函数在程序中设置“日志点”。每当程序运行到“日志点”时，应用程序的日志信息便被发送到 LogCat 中，使用者可以根据“日志点”信息是否与预期的内容一致，判断程序是否存在错误。之所以使用 5 个不同的函数产生日志，主要是为了区分日志信息的类型，其中，Log.v( )用来记录详细信息，Log.d( )用来记录调试信息，Log.i( )用来记录通告信息，Log.w( )用来记录警告信息，Log.e( )用来记录通错误信息。

在下面的程序中，演示了 Log 类的具体使用方法。

```
1    package edu.hrbeu.LogCat;
2
3    import android.app.Activity;
4    import android.os.Bundle;
5    import android.util.Log;
6
7    public class LogCat extends Activity {
8        final static String TAG = "LOGCAT";
9        @Override
10       public void onCreate(Bundle savedInstanceState) {
11           super.onCreate(savedInstanceState);
12           setContentView(R.layout.main);
13
14           Log.v(TAG,"Verbose");
15           Log.d(TAG,"Debug");
16           Log.i(TAG,"Info");
17           Log.w(TAG,"Warn");
18           Log.e(TAG,"Error");
19       }
20   }
```

Problems | Javadoc | Declaration | Console | Properties | LogCat

Log

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 19:38:53.085 | I | 1103 | dalvikvm | Debugger has detached; object registry had 1 entries |
| 06-07 19:38:53.105 | D | 1103 | dalvikvm | VM cleaning up |
| 06-07 19:38:53.255 | D | 1103 | dalvikvm | LinearAlloc 0x0 used 639228 of 4194304 (15%) |
| 06-07 19:38:53.766 | W | 568 | ActivityMa... | Activity pause timeout for HistoryRecord{436711d8 {com.android |
| 06-07 19:38:53.786 | I | 568 | ActivityMa... | Start proc edu.hrbeu.ActivityLifeCycle for activity edu.hrbeu. |
| 06-07 19:38:54.027 | I | 1111 | jdwp | received file descriptor 10 from ADB |
| 06-07 19:38:54.925 | I | 1111 | LIFECYCLE | (1) onCreate() |
| 06-07 19:38:54.945 | I | 1111 | LIFECYCLE | (2) onStart() |
| 06-07 19:38:54.945 | I | 1111 | LIFECYCLE | (4) onResume() |
| 06-07 19:38:55.396 | I | 568 | ActivityMa... | Displayed activity edu.hrbeu.ActivityLifeCycle/.ActivityLifeCy |
| 06-07 19:39:05.406 | D | 611 | dalvikvm | GC freed 602 objects / 33280 bytes in 258ms |

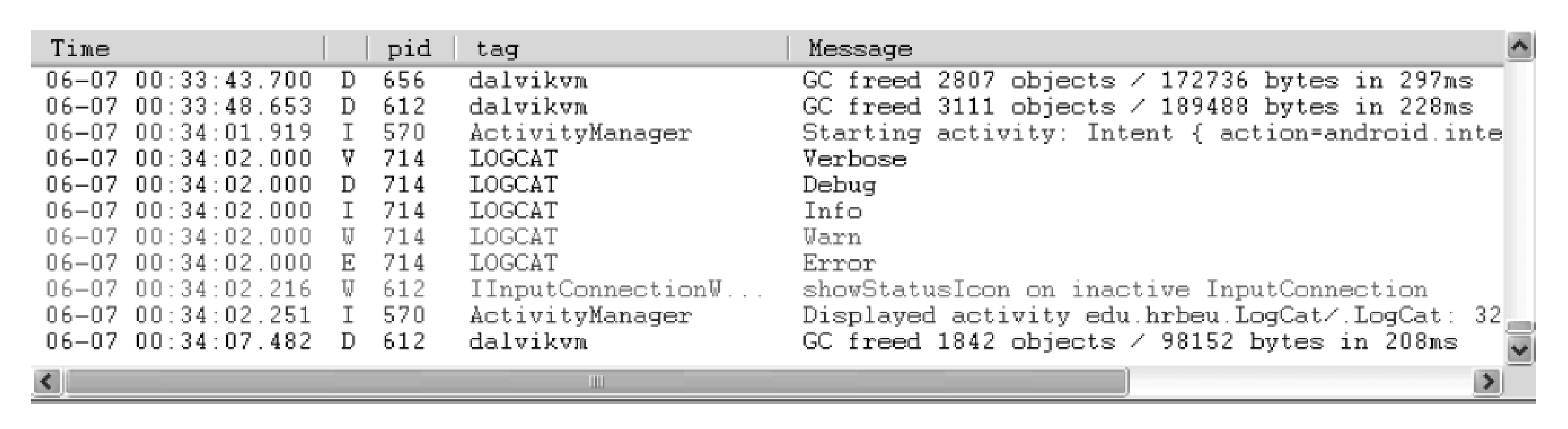
Filter:

**图 4.10 Eclipse 中的 LogCat**

为了使用 Log 类中的函数，首先在程序第 5 行引入 android.util.Log 包；然后在第 8 行定义标签，标签帮助用户在 LogCat 中找到目标程序生成的日志信息，同时也能够利用标签对日志进行过滤；第 14 行记录一个详细信息，Log.v( )函数的第一个参数是日志的标签，第二个参数是实际的信息内容；第 15 行到第 18 行分别产生了调试信息、通告信息、警告信息和错误信息。

程序运行后，LogCat 捕获得到应用程序发送的日志信息，显示结果如图 4.11 所示。在 LogCat 中显示了标签为 LOGCAT 的日志信息共 5 条，并以不同颜色加以显示。可见，LogCat 对不同类型的信息使用了不同的颜色加以区别。

如果能够使用 LogCat 的过滤器，则可以使显示的结果更加清晰。下面使用在 LogCat 右上角的“+”号，添加一个名为 LogcatFilter 的过滤器，并设置过滤条件为“标签＝LOGCAT”，具体设置方法如图 4.12 所示。



| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 00:33:43.700 | D | 656 | dalvikvm | GC freed 2807 objects / 172736 bytes in 297ms |
| 06-07 00:33:48.653 | D | 612 | dalvikvm | GC freed 3111 objects / 189488 bytes in 228ms |
| 06-07 00:34:01.919 | I | 570 | ActivityManager | Starting activity: Intent { action=android.inte |
| 06-07 00:34:02.000 | V | 714 | LOGCAT | Verbose |
| 06-07 00:34:02.000 | D | 714 | LOGCAT | Debug |
| 06-07 00:34:02.000 | I | 714 | LOGCAT | Info |
| 06-07 00:34:02.000 | W | 714 | LOGCAT | Warn |
| 06-07 00:34:02.000 | E | 714 | LOGCAT | Error |
| 06-07 00:34:02.216 | W | 612 | IInputConnectionW... | showStatusIcon on inactive InputConnection |
| 06-07 00:34:02.251 | I | 570 | ActivityManager | Displayed activity edu.hrbeu.LogCat/.LogCat: 32 |
| 06-07 00:34:07.482 | D | 612 | dalvikvm | GC freed 1842 objects / 98152 bytes in 208ms |

**图 4.11　LogCat 工程的运行结果**

Log Filter
Filter Name: LogcatFilter
by Log Tag: LOGCAT
by pid:
by Log level: <none>
OK　Cancel

**图 4.12　LogCat 过滤器**

过滤器设置好后，LogcatFilter 过滤后的日志信息如图 4.13 所示。在这之后，无论什么类型的日志信息，属于哪一个进程，只要标签为 LOGCAT，都将显示在 LogcatFilter 区域内。

Log (3)　LogcatFilter

| Time | | pid | tag | Message |
|---|---|---|---|---|
| 06-07 00:58:18.261 | V | 714 | LOGCAT | Verbose |
| 06-07 00:58:18.274 | D | 714 | LOGCAT | Debug |
| 06-07 00:58:18.279 | I | 714 | LOGCAT | Info |
| 06-07 00:58:18.279 | W | 714 | LOGCAT | Warn |
| 06-07 00:58:18.279 | E | 714 | LOGCAT | Error |

**图 4.13　LogCat 过滤后的输入结果**

## 4.4.2　DevTools

在 Android 模拟器中，内置了一个用于调试和测试的工具 DevTools。DevTools 包括了一系列用户各种用途的小工具，包括 Development Settings、Exception Browser、Google Login Service、Instrumentation、Media Scanner、Package Browser、Pointer Location、Raw Image Viewer、Running processes 和 Terminal Emulator。图 4.14 是 DevTools 的使用界面。

在这些工具中，经常用到的有设置调试选项的 Development Settings，浏览已经安装程序包的 Package Browser，确定触摸点位置的 Pointer Location，查看当前运行进程的 Running processes，还有连接底层 Linux 操作系统的虚拟终端软件 Terminal Emulator。后面的内容将逐一地介绍这些经常使用到小工具的功能和使用方法。

### 1. Development Settings

Development Settings 中包含了程序调试的相关选项，如图 4.15 所示。



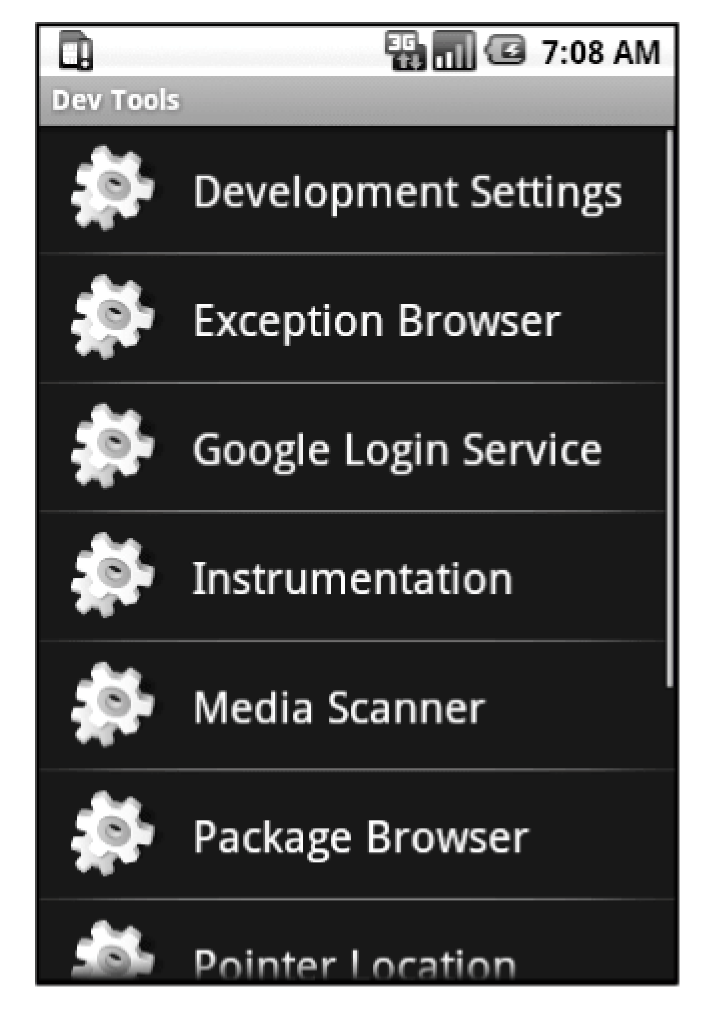




图 4.14 Package Browser

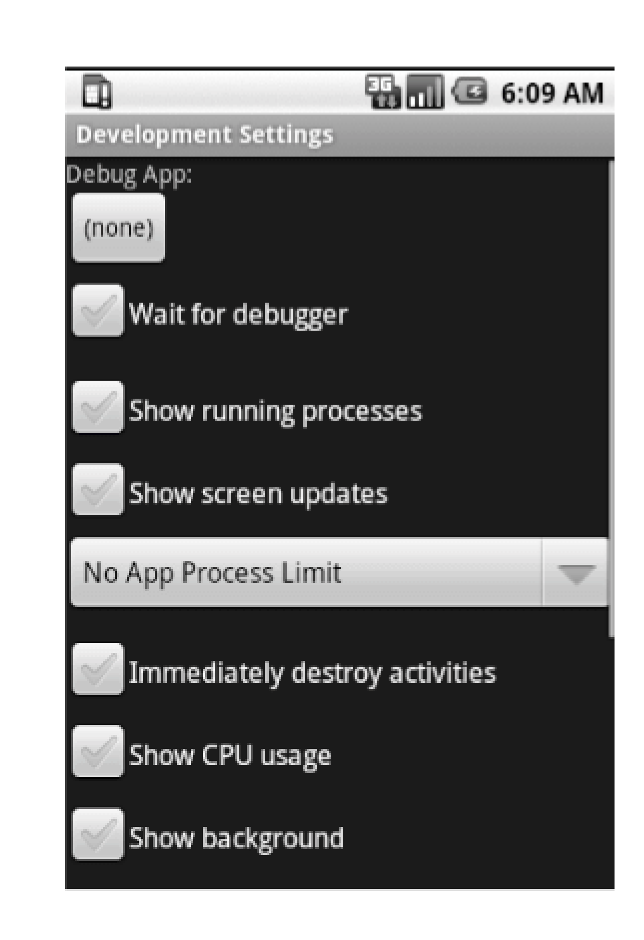



图 4.15 Development Settings

如果希望启动 Development Settings 中某项功能，只需要单击功能前面选择框，出现绿色的"对号"表示功能启用。而且功能启用后，模拟器会自动保存设置，即使再次启动模拟器用户的选择内容仍会存在。Development Settings 每一项的具体说明可以参考表 4.3。

表 4.3 Development Settings 选项

| 选　　项 | 说　　明 |
|---|---|
| Debug App | 为 Wait for debugger 选项指定应用程序，如果不指定（选择 none），Wait for debugger 选项将适用于所有应用程序。Debug App 可以有效地防止 Android 程序长时间停留在断点而产生异常 |
| Wait for debugger | 阻塞加载应用程序，直到关联到调试器（Debugger）。用于在 Activity 的 onCreate()函数进行断点调试 |
| Show running processs | 在屏幕右上角显示运行中的进程 |
| Show screen updates | 选中该选项时，界面上任何被重绘的矩形区域会闪现粉红色，有利于发现界面中不必要的重绘区域 |
| No App Process limit | 允许同时运行进程的数量上限 |
| Immediately destroy activites | Activity 进入停止状态后立即销毁，用于测试在函数 onSaveInstanceState()、onRestoreInstanceState()和 onCreate()中的代码 |
| Show CPU usage | 在屏幕顶端显示 CPU 使用率，上层红线显示总的 CPU 使用率，下层绿线显示当前进程的 CPU 使用率 |
| Show background | 应用程序没有 Activity 显示时，直接显示背景面板，一般这种情况仅在调试时出现 |

续表

| 选　　项 | 说　　明 |
|---|---|
| Show Sleep state on LED | 在休眠状态下开启 LED |
| Windows Animation Scale | |
| Transition Animation | |
| Light Hinting | |
| Show GTalk service connection status | 显示 GTalk 服务连接状态 |

### 2. Package Browser

Package Browser 是 Android 系统中的程序包查看工具,能够详细显示已经安装到 Android 系统中的程序信息,包括包名称、应用程序名称、图标、进程、用户 ID、版本、apk 文件保存位置和数据文件保存位置等,而且能够进一步查看应用程序所包含的 Activity、Service、BroadcastReceiver 和 Provider 的详细信息。图 4.16 是 Package Browser 查看 ActivityLifeCycle 程序的相关信息。

### 3. Pointer Location

Pointer Location 是屏幕点位置查看工具,能够显示触摸点的 X 轴坐标和 Y 轴坐标。图 4.17 是 Pointer Location 的使用画面。



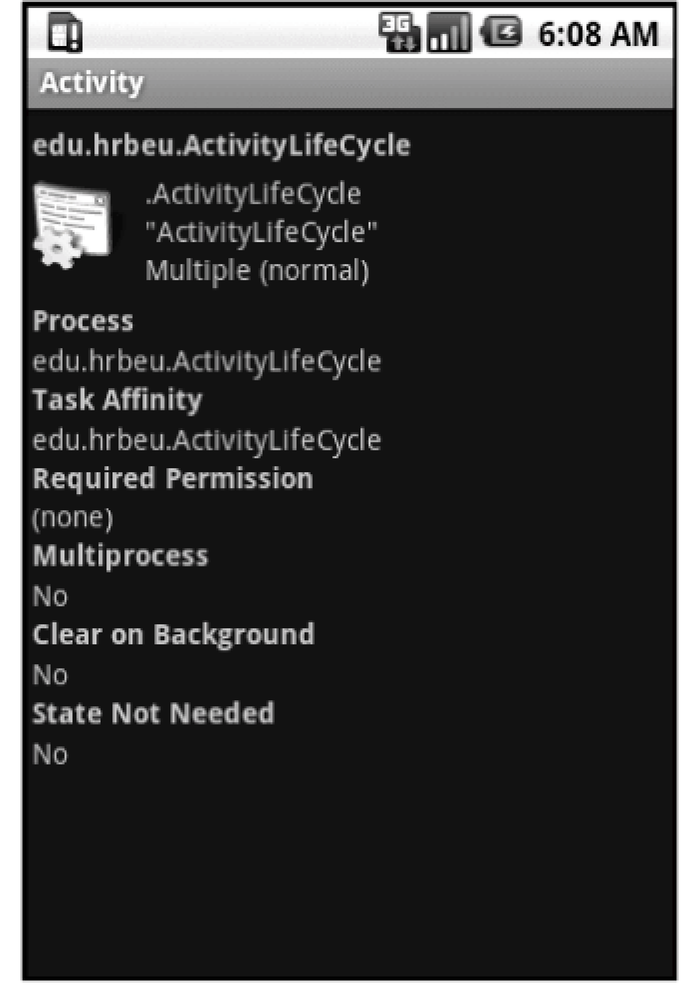




图 4.16 Package Browser

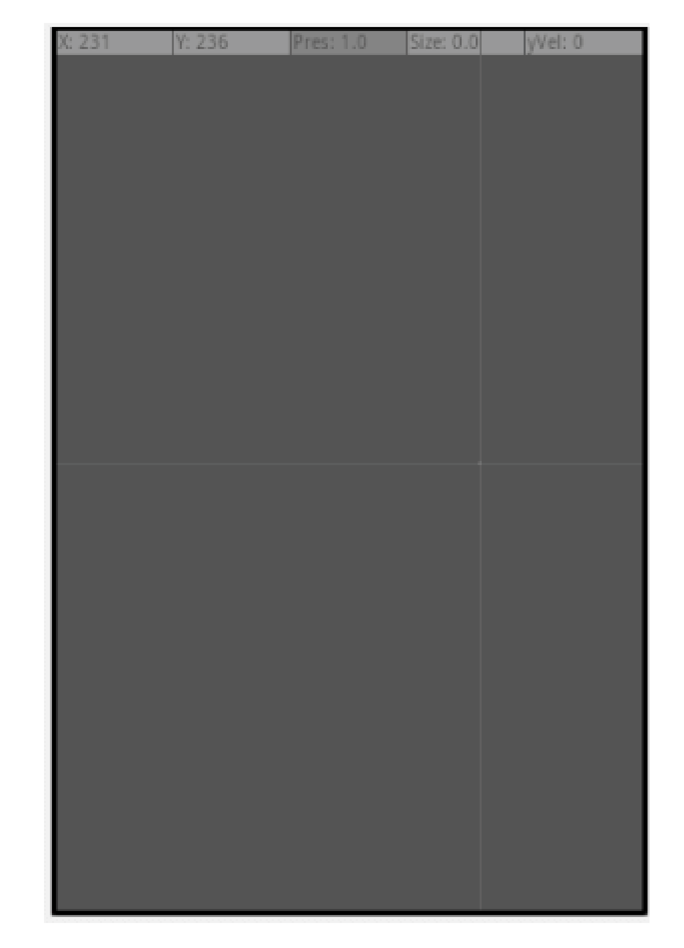


图 4.17 Pointer Location

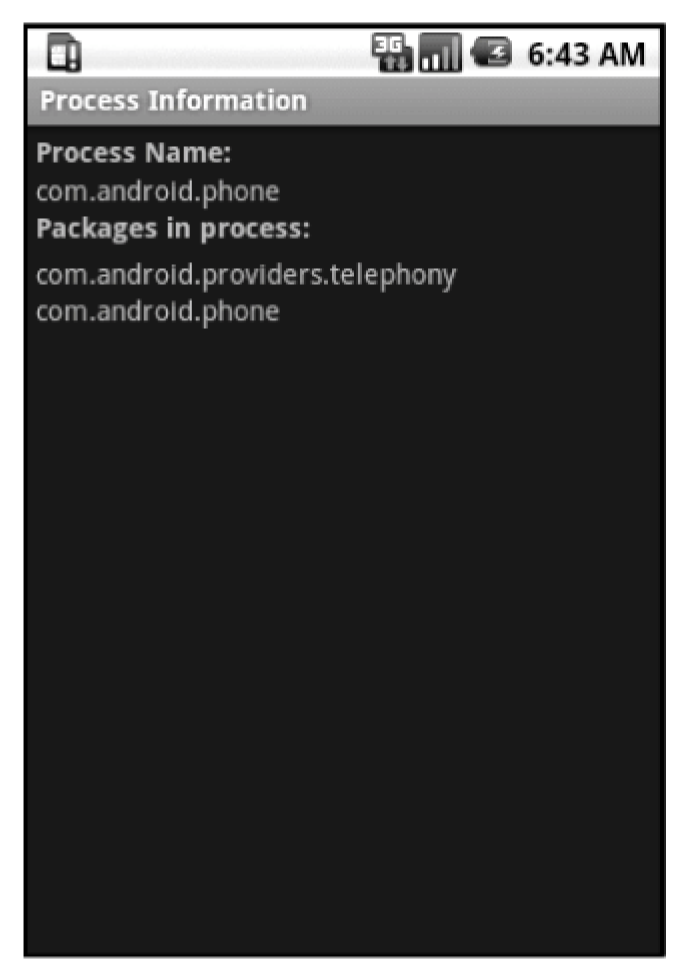
### 4. Running processes

Running processes 能够查看在 Android 系统中正在运行的进程,并能查看进程的详细信息,包括进程名称和进程所调用的程序包。图 4.18 是 Andoird 模拟器默认情况下运行的进程和 com.android.phone 进程的详细信息。

### 5. Terminal Emulator

Terminal Emulator 可以打开一个连接底层 Linux 系统的虚拟终端，但具有的权限较低，且不支持提升权限的 su 命令。如果需要使用 root 权限的命令，可以使用 ADB 工具。图 4.19 是 Terminal Emulator 运行时的画面，输入 ls 命令，显示出根目录下的所有文件夹。



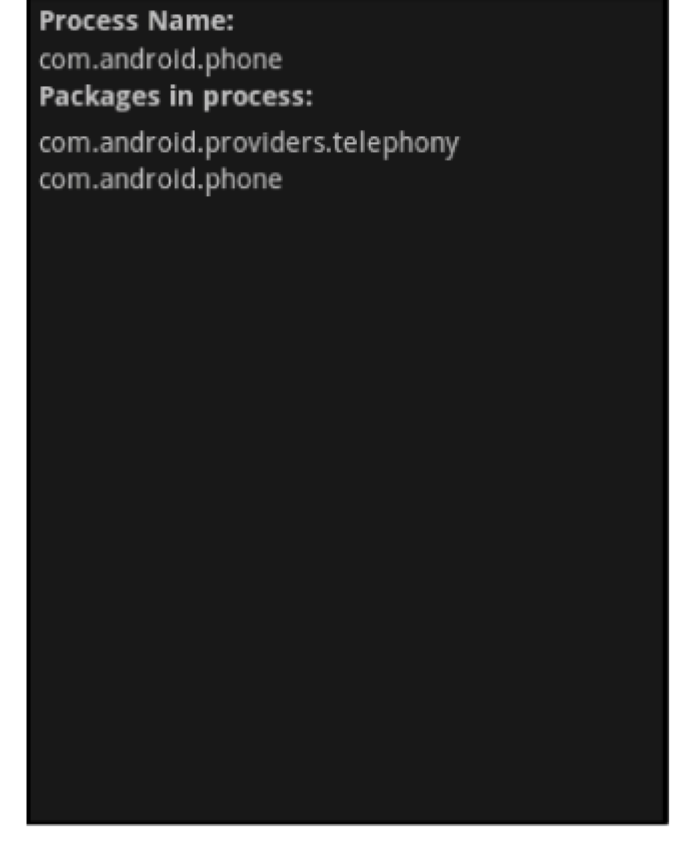



图 4.18 Running processes




图 4.19 Terminal Emulator

## 习 题

1. 简述 Android 系统前台进程、可见进程、服务进程、后台进程和空进程的优先级排序原因。

2. 简述 Android 系统的 4 种基本组件 Activity、Service、BroadcaseReceiver 和 ContentProvider 的用途。

3. 简述 Activity 生命周期的 4 种状态，以及状态之间的变换关系。

4. 简述 Activity 事件回调函数的作用和调用顺序。

# 第 5 章 chapter 5

# Android 用户界面

用户界面是应用程序开发的重要组成部分，决定了应用程序是否美观、易用。通过本章的学习可以让读者熟悉 Android 用户界面的基本开发方法，了解在 Android 界面开发过程中常见的界面控件、界面布局、菜单和界面事件的使用方法，充分理解手机应用程序与桌面应用程序在用户界面开发上的不同与相同之处。

**本章学习目标：**

- 了解各种用户界面的控件的使用方法
- 掌握各种界面布局的特点和使用方法
- 掌握选项菜单、子菜单和快捷菜单的使用方法
- 掌握按键事件和触摸事件的处理方法

## 5.1 用户界面基础

用户界面（User Interface，UI）是系统和用户之间进行信息交换的媒介，实现信息的内部形式与人类可以接受形式之间的转换。最古老的用户界面是各种形式的文字、图形、旗帜和手势等，这些抽象符号作为信息传递的介质，是人类可以理解这些信息所包含的意义或指代的实体。

算盘是由珠子组成的最早的人机交互界面。在计算机出现的早期，批处理界面（1945—1968）和命令行界面（1969—1983）得到广泛的使用。目前，流行的用户界面是图像用户界面（Graphical User Interface，GUI），采用图形方式与用户进行交互。与早期的交互界面相比，图形界面对于用户来说更加简便易用，用户从此不再需要记住大量的命令，取而代之的是通过窗口、菜单和按钮等方式来进行操作。未来的用户界面将更多地运用虚拟现实技术，使用户能够摆脱键盘与鼠标的交互方式，而通过动作、语言，甚至是脑电波来控制计算机。当然，对用户界面的深入探讨远超出了本书的涉及范围，感兴趣的读者可以阅读相关方面的书籍。

在手机上进行用户界面设计是一项具体挑战性的工作。首先，手机界面的设计者和程序开发人员一般是独立且并行工作的，这就需要界面设计与程序逻辑完全分离，这样不仅有利于并行开发，而且在后期修改界面时，也不用再次修改程序的逻辑代码；其次，不同型号手机的屏幕解析度、尺寸和纵横比各不相同，用户界面需要能够根据屏幕信息，

自动调整界面上部分控件的位置和尺寸，避免因为屏幕解析度、尺寸或纵横比的变化而出现显示错误；最后，手机屏幕尺寸较小，设计者必须能够合理利用有限的显示空间，构造出符合人机交互规律的用户界面，避免出现凌乱、拥挤的用户界面。

Android 系统已经为使用者解决了界面设计前两个问题。在界面设计与程序逻辑分离方面，Android 程序将用户界面和资源从逻辑代码中分离出来，使用 XML 文件对用户界面进行描述，资源文件独立保存在资源文件夹中。Android 系统的界用户面描述非常灵活，允许不明确定义界面元素的位置和尺寸，仅声明界面元素的相对位置和粗略尺寸，从而使界面元素能够根据屏幕尺寸和屏幕摆放方式动态调整显示方式。

Android 用户界面框架（Android UI Framework）采用 MVC（Model-View-Controller）模型，如图 5.1 所示。MVC 模型为用户界面提供了处理用户输入的控制器（Controller），显示用户界面和图像的视图（View），以及保存数据和代码的模型（Model）。

MVC 模型中的视图将应用程序的信息反馈给用户，可能的反馈方法包括视觉、听觉或触觉等，但最常用的就是通过屏幕显示反馈信息。Android 用户界面框架中的界面元素以一种树形结构组织在一起，称为视图树（View Tree），如图 5.2 所示。Android 系统在屏幕上绘制界面元素时，会依据视图树的结构从上至下绘制每一个界面元素。每个元素负责对自身的绘制，如果元素包含子元素，该元素会通知其下所有子元素进行绘制。Android 系统在用户界面绘制上还有着一些提高效率的方法，例如如果父元素能够确定某个区域一定会被其子元素绘制，则父元素会停止绘制该区域，以提高屏幕绘制的效率，缩短绘制时间。

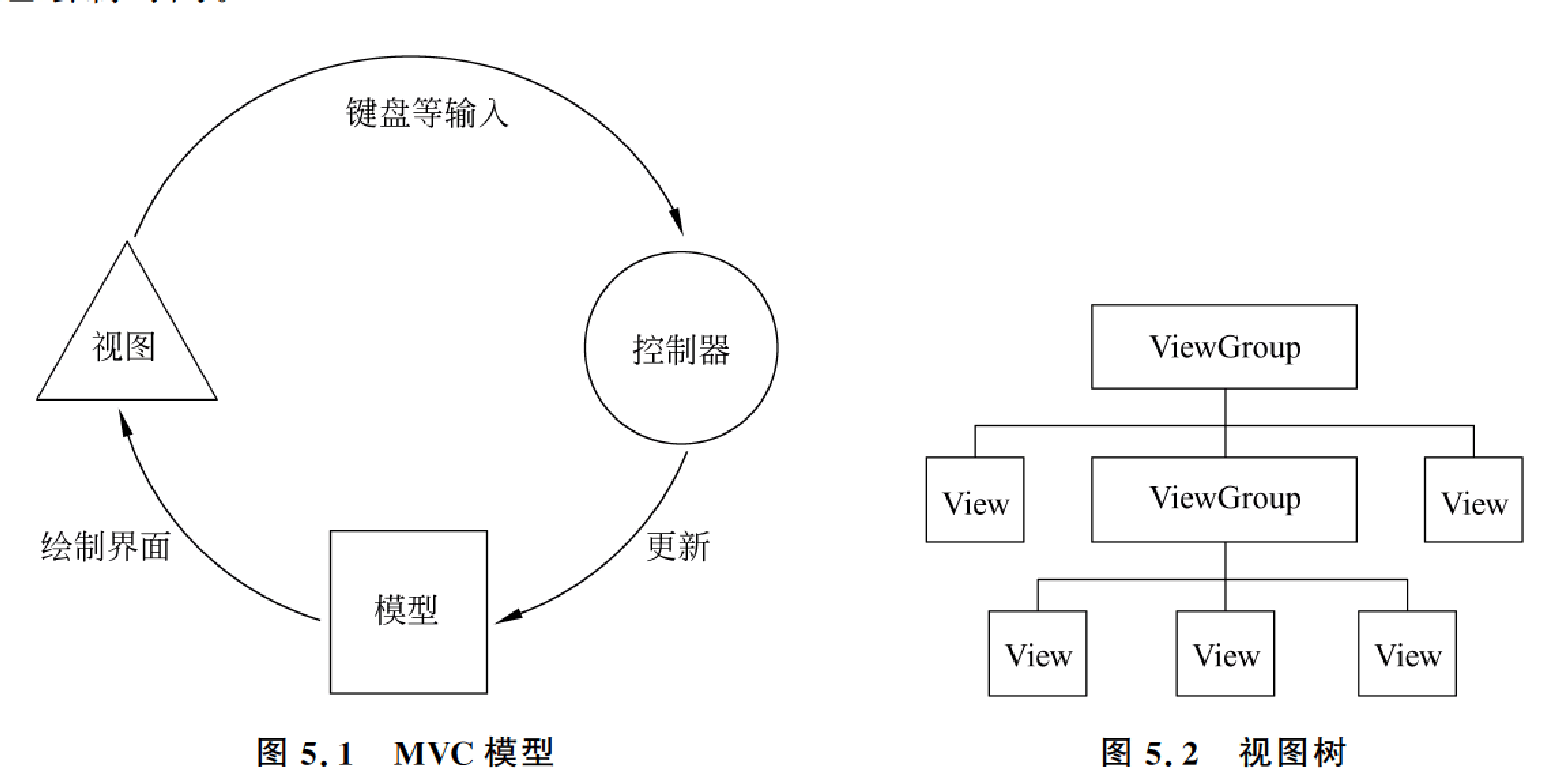



**图 5.1 MVC 模型**　　**图 5.2 视图树**

视图树由 View 和 ViewGroup 构成。View 是界面的最基本的可视单元，存储了屏幕上特定矩形区域内所显示内容的数据结构，并能够实现所占据区域的界面绘制、焦点变化、用户输入和界面事件处理等功能。View 也是一个重要的基类，所有在界面上的可见元素都是 View 的子类。ViewGroup 是一种能够承载含多个 View 的显示单元，一般有两个用途，一个是承载界面布局，另一个是承载具有原子特性的重构模块。

MVC 模型中的控制器能够接受并响应程序的外部动作，如按键动作或触摸屏动作

等。控制器使用队列处理外部动作，每个外部动作作为一个对立的事件被加入队列中，然后 Android 用户界面框架按照“先进先出”的规则从队列中获取事件，并将这个事件分配给所对应的事件处理函数。

Android 用户界面框架中另一个重要的概念就是单线程用户界面(Single-threaded UI)。在单线程用户界面中，控制器从队列中获取事件和视图在屏幕上绘制用户界面，使用的都是同一个线程。单线程用户界面能够降低应用程序的复杂程度，同时也能降低开发的难度。首先，用户不需要在控制器和视图之间进行同步。其次，所有事件处理完全按照其加入队列的顺序进行，也就是说，在事件处理函数返回前不会处理其他的事件，因此用户界面的事件处理函数具有原子性。但单线程用户界面也有它的缺点，如果事件处理函数过于复杂，可能会导致用户界面失去响应。因此尽可能在事件处理函数中使用简短的代码，或将复杂的工作交给后台线程处理。

## 5.2 界面控件

Android 系统的界面控件分为定制控件和系统控件。定制控件是用户独立开发的控件，或通过继承并修改系统控件后所产生的新控件，能够为用户提供特殊的功能或与众不同的显示需求方式。系统控件 Android 系统提供给用户已经封装的界面控件，提供在应用程序开发过程中常见功能控件。系统控件更有利于帮助用户进行快速开发，同时能够使 Android 系统中应用程序的界面保持一致性。

常见的系统控件包括 TextView、EditText、Button、ImageButton、CheckBox、RadioButton、Spinner、ListView 和 TabHost。

### 5.2.1 TextView和 EditText

TextView 是一种用于显示字符串的控件，EditText 则是用来输入和编辑字符串的控件，因为 EditText 继承于 TextView，所以 EditText 是一个具有编辑功能的 TextView。TextViewDemo 是一个同时具有 TextView 和 EditText 的示例，如图 5.3 所示，上方“用户名”部分使用的是 TextView，下方的文字输入框使用的是 EditText。

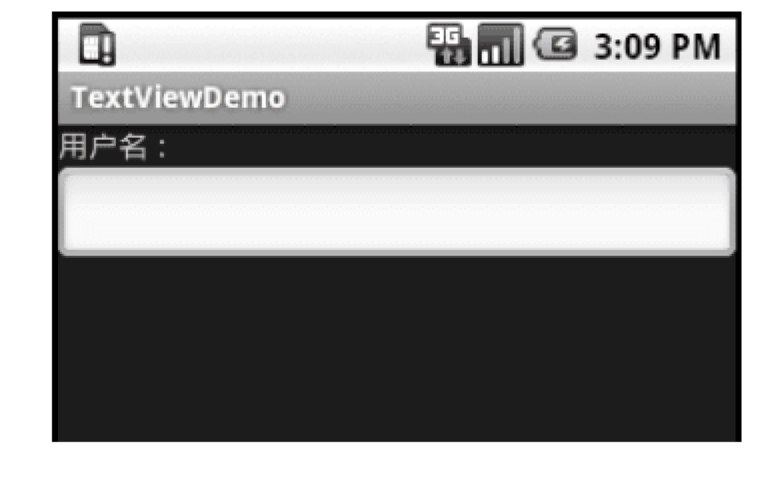



图 5.3 TextView 和 EditText

TextView 和 EditText 控件的命名分别是 TextView01 和 EditText01，在 XML 文件中(/res/layout/main.xml)中的代码如下：

```
<TextView android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView01">
</TextView>
<EditText android:id="@+id/EditText01"
    android:layout_width="fill_parent"
```

```
8         android:layout_height="wrap_content"
9         android:text="EditText01">
10    </EditText>
```

第 1 行 android:id 属性声明了 TextView 的 ID，这个 ID 主要用于在代码中引用这个 TextView 对象。“@+id/TextView01”表示所设置的 ID 值，其中@表示后面的字符串是 ID 资源；加号(+)表示需要建立新资源名称，并添加到 R.java 文件中；斜杠后面的字符串(TextView01)表示新资源的名称。如果属于不是新添加的，或属于 Android 框架的 ID 资源，则不需要使用加号(+)，但必须添加 Android 包的命名空间，例如 android:id="@android:id/empty"。

第 2 行的 android:layout_width 属性用来设置 TextView 的宽度，wrap_content 表示 TextView 的宽度只要能够包含所显示的字符串即可。第 3 行的 android:layout_height 属性用来设置 TextView 的高度。第 4 行表示 TextView 所显示的字符串，在后面将通过代码更改 TextView 的显示内容。第 7 行中 fill_content 表示 EditText 的宽度将等于父控件的宽度。

TextViewDemo.java 文件中，引用 XML 文件中建立的 TextView 和 EditText，并更改其显示内容。为了能够使程序正常运行，需要在代码中引入 android.widget.EditText 和 android.widget.TextView。

```
1         TextView textView= (TextView)findViewById(R.id.TextView01);
2         EditText editText= (EditText)findViewById(R.id.EditText01);
3         textView.setText("用户名：");
4         editText.setText("");
```

第 1 行的 findViewById()函数能够通过 ID 引用界面上的任何控件，只要该控件在 XML 文件中定义过 ID 即可。第 3 行的 setText()函数用来设置 TextView 所显示的内容。

### 5.2.2 Button 和 ImageButton

Button 是一种按钮控件，用户能够在该控件上点击，并后引发相应的事件处理函数。很多情况下，用户需要一个能够显示图像的按钮，Android 系统提供了 ImageButton 控件，用以实现能够显示图像功能的按钮。ButtonDemo 是一个简单的按钮示例，如图 5.4 所示，上方的“Button 按钮”是一个 Button 控件，下方的按钮是一个 ImageButton 控件。

ButtonDemo 示例从上至下的控件 ID 分别是 TextView01、Button01 和 ImageButton01。在 XML 文件中(/res/layout/main.xml)的代码如下：

```
<Button android:id="@+id/Button01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Button01">
</Button>
<ImageButton android:id="@+id/ImageButton01"
      android:layout_width="wrap_content"
```

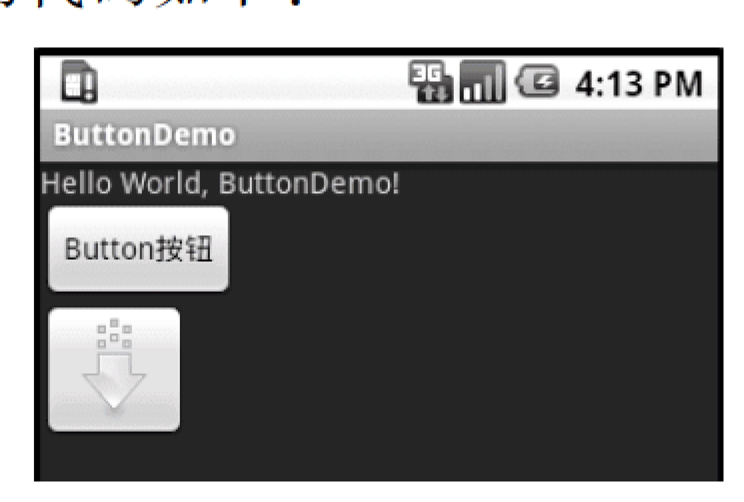


**图 5.4 Button 和 ImageButton**

```
        android:layout_height="wrap_content">
    </ImageButton>
```

XML 文件中定义了两个按钮的宽度和高度，并定义了 Button 控件所显示的内容，但没有定义 ImageButton 所显示的图像，所显示的图像在后面的代码中进行定义。

Android 系统支持多中图形格式，例如 png、ico 等等，本例中使用 png 文件作为 ImageButton 的显示图像。首先将 download. png 文件拷贝到/res/drawable 文件夹下，然后在/res 目录上选择 Refresh(见图 5.5)，刷新后新添加的文件将显示在/res/drawable 文件夹下，同时 R. java 文件内容也得到了更新，否则无法在代码中使用该资源，会提示无法找到资源的错误。

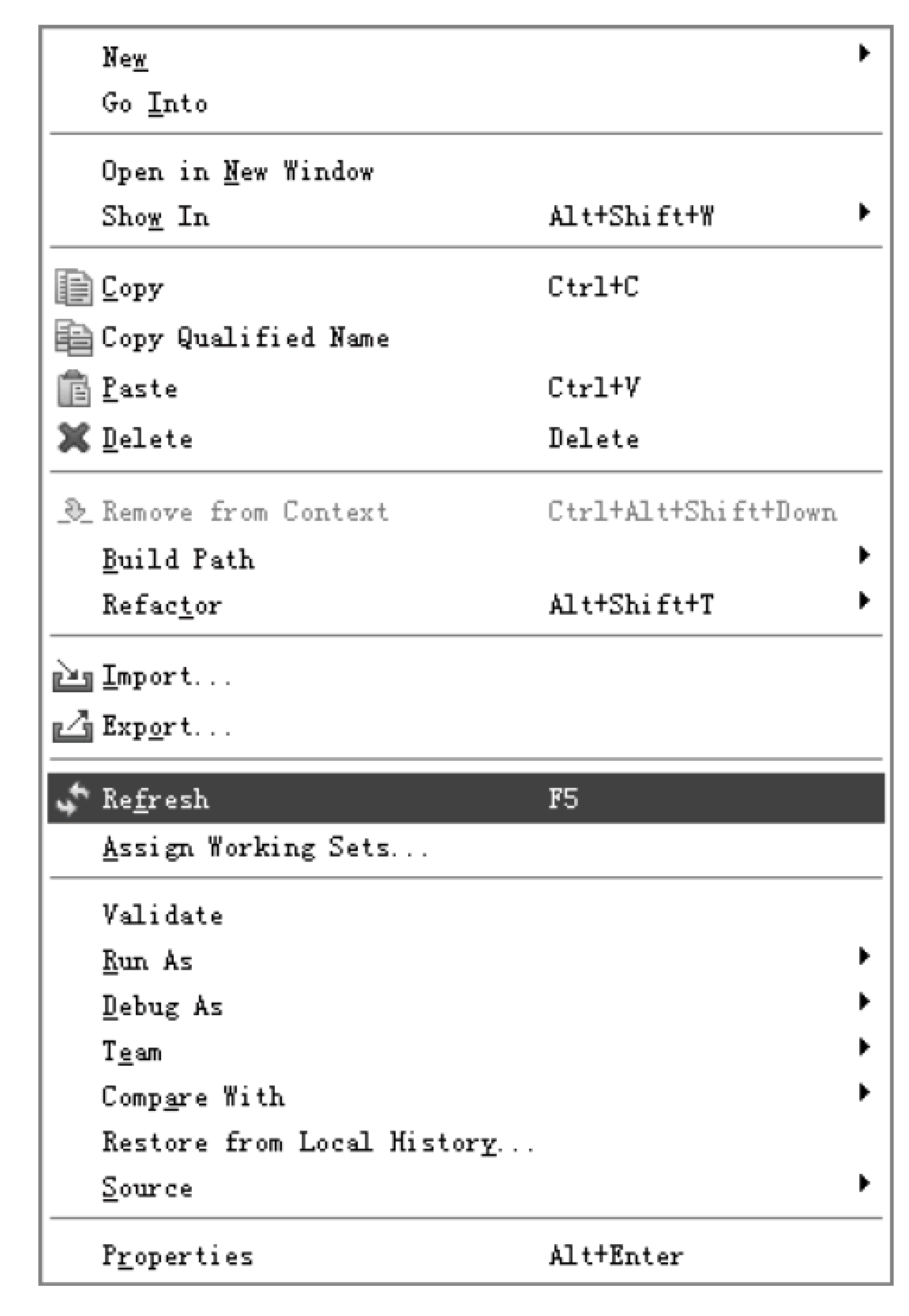



**图 5.5 更新 R. java 文件**

ButtonDemo. java 文件中，引用 XML 文件中建立的 Button 和 ImageButton，更改 Button 的显示字符内容和 ImageButton 图像内容。为了能够使程序正常运行，需要在代码中引入 android. widget. Button 和 android. widget. ImageButton。

```
1    Button button= (Button)findViewById(R.id.Button01);
2    ImageButton imageButton= (ImageButton)findViewById(R.id.ImageButton01);
3    button.setText("Button 按钮");
4    imageButton.setImageResource(R.drawable.download);
```

第 1 行和第 2 行代码用于引用在 XML 文件中定义的 Button 控件和 ImageButton 控件。第 3 行代码将 Button 的显示内容更改为“Button 按钮”。第 4 行代码利用 setImageResource( ) 函数，将新加入的 png 文件 R. drawable. download 传递给 ImageButton。

为了能够使按钮响应点击事件，在 onCreate()函数中为 Button 控件和 ImageButton 控件添加点击事件的监听器，代码如下：

```
        final TextView textView= (TextView)findViewById(R.id.TextView01);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                textView.setText("Button 按钮");
            }
        });
        imageButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                textView.setText("ImageButton 按钮");
```

```
10            }
11        });
```

在第 2 行代码中，button 对象通过调用 setOnClickListener()函数，注册一个点击(Click)事件的监听器 View. OnClickListener()。第 3 行代码是点击事件的回调函数。第 4 行代码将 TextView 的显示内容更改为“Button 按钮”。

View. OnClickListener()是 View 定义的点击事件的监听器接口，并在接口中仅定义了 onClick()函数。当 Button 从 Android 界面框架中接收到事件后，首先检查这个事件是否是点击事件，如果是点击事件，同时 Button 又注册了监听器，则会调用该监听器中的 onClick()函数。

每个 View 仅可以注册一个点击事件的监听器，如果使用 setOnClickListener()函数注册第二个点击事件的监听器，之前注册的监听器将被自动注销。给每个按钮注册一个点击事件监听器，每个按钮的事件处理程序都在各自的 onClick()函数，这样能够使代码更加清晰、易读，且易于维护。当然，也可以将多个按钮注册到同一个点击事件的监听器上，示例代码如下：

```
1   Button.OnClickListener buttonListener= new Button.OnClickListener(){
2           @Override
3           public void onClick(View v) {
4               switch(v.getId()){
5                   case R.id.Button01:
6                       textView.setText("Button 按钮");
7                       return;
8                   case R.id.ImageButton01:
9                       textView.setText("ImageButton 按钮");
10                      return;
11              }
12          }};
13      button.setOnClickListener(buttonListener);
14      imageButton.setOnClickListener(buttonListener);
```

第 1 行至第 12 行代码定义了一个名为 buttonListener 的点击事件监听器，第 13 行和第 14 行代码分别将给该监听器注册到 Button 和 ImageButton 上。

### 5.2.3 CheckBox 和 RadioButton

CheckBox 是一个同时可以选择多个选项的控件，而 RadioButton 则是仅可以选择一个选项的控件。RadioGroup 是 RadioButton 的承载体，程序运行时不可见，应用程序中可能包含一个或多个 RadioGroup。RadioGroup 包含多个 RadioButton，在一个 RadioGroup 中，用户仅能够选择其中一个 RadioButton。在图 5. 6 中，RadioGroup 中包含两个 RadioButton，当选择 Radio Button1 后，RadioButton2 自动变为非选择状态。

CheckboxRadiobuttonDemo 示例从上至下的控件 ID 分别是 TextView01、CheckBox01、CheckBox02、RadioButton01 和 RadioButton02。在 XML 文件中(/res/

layout/main.xml)的代码如下：

```
1   <TextView android:id="@+id/TextView01"
2       android:layout_width="fill_parent"
3       android:layout_height="wrap_content"
4       android:text="@string/hello"/>
5   <CheckBox android:id="@+id/CheckBox01"
6       android:layout_width="wrap_content"
7       android:layout_height="wrap_content"
8       android:text="CheckBox01">
9   </CheckBox>
10  <CheckBox android:id="@+id/CheckBox02"
11      android:layout_width="wrap_content"
12      android:layout_height="wrap_content"
13      android:text="CheckBox02">
14  </CheckBox>
15  <RadioGroup android:id="@+id/RadioGroup01"
16      android:layout_width="wrap_content"
17      android:layout_height="wrap_content">
18      <RadioButton android:id="@+id/RadioButton01"
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="RadioButton01">
22      </RadioButton>
23      <RadioButton android:id="@+id/RadioButton02"
24          android:layout_width="wrap_content"
25          android:layout_height="wrap_content"
26          android:text="RadioButton02">
27      </RadioButton>
28  </RadioGroup>
```




**图 5.6　CheckBox 和 RadioButton**

第 15 行<RadioGroup>标签声明了一个 RadioGroup，在第 18 行和第 23 行分别声明了两个 RadioButton，这两个 RadioButton 是 RadioGroup 的子元素。

在代码中引用 CheckBox 和 RadioButton 的方法可以参考下面的代码：

```
CheckBox checkBox1= (CheckBox)findViewById(R.id.CheckBox01);
RadioButton radioButton1= (RadioButton)findViewById(R.id.RadioButton01);
```

CheckBox 设置点击事件监听器的方法，与 Button 中介绍的方法相似，唯一不同在于将 Button.OnClickListener 换成了 CheckBox.OnClickListener，下面给出简要代码：

```
CheckBox.OnClickListener checkboxListener=new CheckBox.OnClickListener(){
    @Override
    public void onClick(View v) {
        //过程代码
    }};
```

```
checkBox1.setOnClickListener(checkboxListener);
checkBox2.setOnClickListener(checkboxListener);
```

RadioButton 设置点击事件监听器的方法：

```
RadioButton.OnClickListener radioButtonListener=new RadioButton.OnClickListener(){
    @Override
    public void onClick(View v) {
      //过程代码
    }};
radioButton1.setOnClickListener(radioButtonListener);
radioButton2.setOnClickListener(radioButtonListener);
```

## 5.2.4 Spinner

Spinner 是一种能够从多个选项中选择一个选项的控件，类似于桌面程序的组合框(ComboBox)，但没有组合框的下拉菜单，而是使用浮动菜单为用户提供选择。SpinnerDemo 是一个包含 3 个子项的 Spinner 控件示例，如图 5.7 所示。




图 5.7 Spinner

SpinnerDemo 示例从上至下的控件 ID 分别为 TextView01 和 Spinner01。在 XML 文件(/res/layout/main.xml)中的代码如下：

```
1  <TextView   android:id="@+id/TextView01"
2              android:layout_width="fill_parent"
3              android:layout_height="wrap_content"
4              android:text="@string/hello"/>
5  <Spinner android:id="@+id/Spinner01"
6        android:layout_width="300dip"
7        android:layout_height="wrap_content">
8  </Spinner>
```

第 5 行使用<Spinner>标签声明了一个 Spinner 控件，并在第 6 行代码中指定了该控件的宽度为 300dip。

在 SpinnerDemo.java 文件中，定义一个 ArrayAdapter 适配器，在 ArrayAdapter 中添加需要在 Spinner 中可以选择的内容。为了能够使程序正常运行，需要在代码中引入 android.widget.ArrayAdapter 和 android.widget.Spinner。

```
Spinner spinner= (Spinner) findViewById(R.id.Spinner01);
List<String> list  =new ArrayList<String>();
list .add("Spinner 子项 1");
list .add("Spinner 子项 2");
list .add("Spinner 子项 3");
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,
```

```
            android.R.layout.simple_spinner_item, list);
7    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
8    spinner.setAdapter(adapter);
```

第2行代码建立了一个字符串数组列表(ArrayList),这种数组列表可以根据需要进行增减,<String>表示数组列表中保存的是字符串类型的数据。在代码的第3,4,5行中,使用add()函数分别向数组列表中添加3个字符串。第6行代码建立了一个ArrayAdapter的数组适配器,数组适配器能够将界面控件和底层数据绑定在一起。ArrayAdapter将Spinner和ArrayList绑定在一起,所有ArrayList中的数据,将显示在Spinner的浮动菜单中,绑定过程由第8行代码实现。第7行代码设定了Spinner的浮动菜单的显示方式,其中,android.R.layout.simple_spinner_dropdown_item是Android系统内置的一种浮动菜单,如图5.7所示。如果将其改为android.R.layout.simple_spinner_item,则显示效果如图5.8所示。

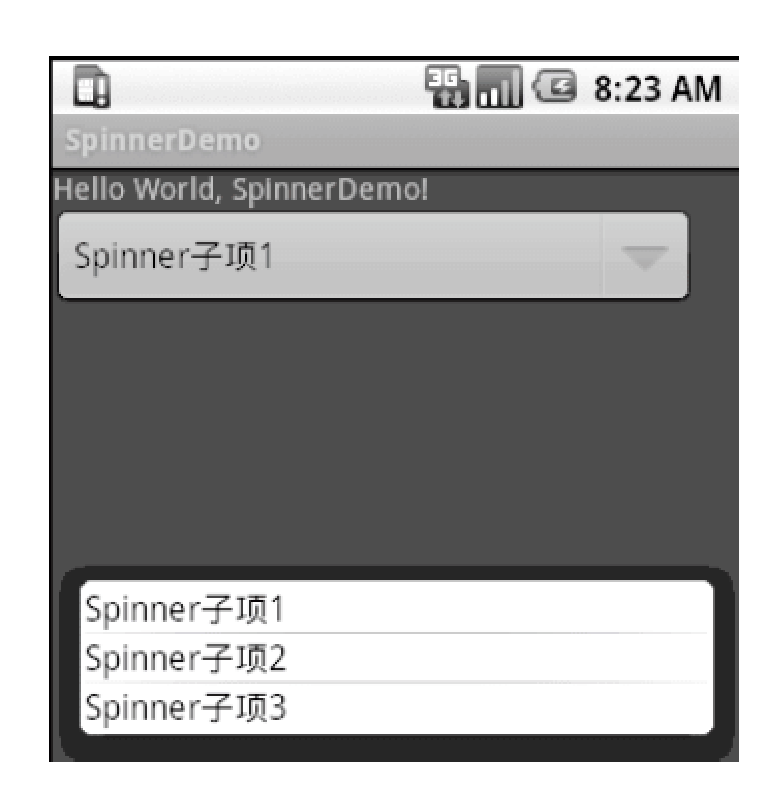



**图5.8 Spinner的item菜单**

为了保证用户界面的显示内容与底层数据的一致性,应用程序需要监视底层数据的变化,如果底层数据更改了,则用户界面也需要修改显示内容。在使用适配器绑定界面控件和底层数据后,应用程序就不需要再监视底层数据的变化,从而极大地简化了代码的复杂性。

### 5.2.5 ListView

ListView是一种用于垂直显示的列表控件,如果显示内容过多,则会出现垂直滚动条。ListView是在界面设计中经常使用的界面控件,主要是因为ListView能够通过适配器将数据和自身绑定,在有限的屏幕上提供大量内容供用户选择;而且支持点击事件处理,用户可以用少量的代码实现复杂的选择功能。ListViewDemo是一个用来说明如何使用ListView的示例,界面如图5.9所示。

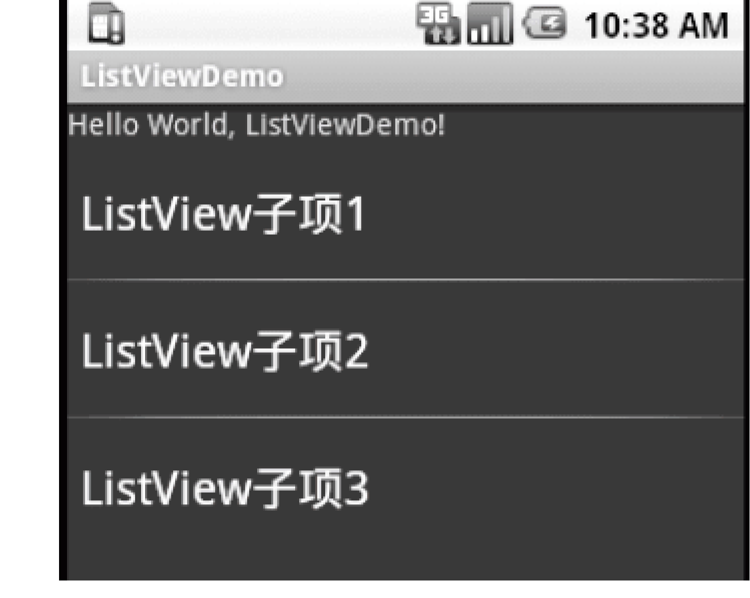



**图5.9 ListView**

ListViewDemo示例从上至下的控件ID分别为TextView01和ListView01。在XML文件(/res/layout/main.xml)中的代码如下:

```
<TextView  android:id="@+id/TextView01"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="@string/hello"/>
<ListView android:id="@+id/ListView01"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content">
</ListView>
```

在ListViewDemo.java文件中,首先需要为ListView创建适配器,并添加ListView

中所显示的内容。

```
1    final TextView textView= (TextView)findViewById(R.id.TextView01);
2    ListView listView= (ListView)findViewById(R.id.ListView01);
3    List< String> list   =new ArrayList< String> ();
4    list.add("ListView 子项 1");
5    list.add("ListView 子项 2");
6    list.add("ListView 子项 3");
7    ArrayAdapter< String> adapter= new ArrayAdapter< String> (this,
             android.R.layout.simple_list_item_1, list );
8    listView.setAdapter(adapter);
```

第 2 行代码通过 ID 引用了 XML 文件中声明的 ListView。第 3 行至第 6 行声明了数组列表，第 7 行声明了适配器 ArrayAdapter，第三个参数 list 说明适配器的数据源为数组列表。第 8 行将 ListView 和适配器绑定。

下面的代码声明了 ListView 子项的点击事件监听器，用以确定用户在 ListView 中，选择的是哪一个子项。

```
1    AdapterView.OnItemClickListener listViewListener= new
             AdapterView.OnItemClickListener(){
2        @Override
3        public void onItemClick(AdapterView< ?> arg0, View arg1, int arg2, long arg3) {
4            String msg = "";
5            textView.setText(msg);
6        }};
7    listView.setOnItemClickListener(listViewListener);
```

第 1 行的 AdapterView.OnItemClickListener 是 ListView 子项的点击事件监听器，同样是一个接口，需要实现 onItemClick()函数。在 ListView 子项被选择后，onItemClick()函数将被调用。第 3 行的 onItemClick()函数中一共有 4 个参数，参数 0 表示适配器控件，就是 ListView；参数 1 表示适配器内部的控件，是 ListView 中的子项；参数 2 表示适配器内部的控件，也就是子项的位置；参数 3 表示子项的行号。第 4 行和第 5 行代码用于显示信息，选择子项确定后，在 TextView 中显示子项父控件的信息、子控件信息、位置信息和 ID 信息。第 7 行代码是 ListView 指定刚刚声明的监听器。

### 5.2.6 TabHost

Tab 标签页是界面设计时经常使用的界面控件，可以实现多个分页之间的快速切换，每个分页可以显示不同的内容。图 5.10 所示是 Android 系统内置的 Tab 标签页，单击“呼出/接听键”后出现，用于电话呼出和查看拨号记录、联系人。

Tab 标签页的使用首先要设计所有的分页的界面布局，在分页设计完成后，使用代码建立 Tab 标签页，并给每个分页添加标识和标题，最后确定每个分页所显示的界面

布局。

在设计分页的界面布局时，使用的方法与设计普通用户界面没有什么区别。为了便于可视化和编码，为每个分页建立一个 XML 文件，用以编辑和保存分页的界面布局。在 TabDemo 示例中，在/res/layout 目录下建立 3 个 XML 文件，分别为 tab1.xml、tab2.xml 和 tab3.xml，这 3 个文件分别使用线性布局、相对布局和绝对布局示例中的 main.xml 的代码，并将布局的 ID 分别定义为 layout01、layout02 和 layout03。下面分别给出 tab1.xml、tab2.xml 和 tab3.xml 文件的部分代码。

**图 5.10　TabHost**

tab1.xml 文件代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout android:id="@+id/layout01"
  …
  …
</LinearLayout>
```

tab2.xml 文件代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<AbsoluteLayout android:id="@+id/layout02"
  …
  …
</AbsoluteLayout>
```

tab3.xml 文件代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<RelativeLayout android:id="@+id/layout03"
  …
  …
</RelativeLayout>
```

分页的布局代码设计完成后，在 TabDemo.java 文件中输入下面的代码，创建 Tab 标签页，并建立子页与界面布局直接的关联关系。

```
package edu.hrbeu.TabDemo;

import android.app.TabActivity;
import android.os.Bundle;
import android.widget.TabHost;
import android.view.LayoutInflater;

public class TabDemo extends TabActivity {
```

```
9       @Override
10      public void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          TabHost tabHost=getTabHost();
13          LayoutInflater.from(this).inflate(R.layout.tab1,
                tabHost.getTabContentView(),true);
14          LayoutInflater.from(this).inflate(R.layout.tab2,
                tabHost.getTabContentView(),true);
15          LayoutInflater.from(this).inflate(R.layout.tab3,
                tabHost.getTabContentView(),true);
16          tabHost.addTab(tabHost.newTabSpec("TAB1")
17              .setIndicator("线性布局").setContent(R.id.layout01));
18          tabHost.addTab(tabHost.newTabSpec("TAB2")
19              .setIndicator("绝对布局").setContent(R.id.layout02));
20          tabHost.addTab(tabHost.newTabSpec("TAB3")
21              .setIndicator("相对布局").setContent(R.id.layout03));
22      }
23  }
```

第 8 行代码的声明 TabDemo 类继承于 TabActivity，与以往继承 Activity 不同，TabActivity 支持内嵌多个 Activity 或 View。第 12 行代码通过 getTabHost()函数获得了 Tab 标签页的容器，用以承载可以点击的 Tab 标签和分页的界面布局。第 13 行代码通过 LayoutInflater 将 tab1. xml 文件中的布局转换为 Tab 标签页可以使用的 View 对象。第 16 行代码使用 addTab()函数添加了第 1 个分页，tabHost. newTabSpec("TAB1")表明在第 12 行代码中建立的 tabHost 上，添加一个标识为 TAB1 的 Tab 分页。第 17 行代码使用 setIndicator()函数设定分页显示的标题，使用 setContent()函数设定分页所关联的界面布局。TabDemo 示例的运行结果如图 5.11 所示。





(a) “线性布局” 分页




(b) “绝对布局” 分页

**图 5.11　TabDemo 示例的运行结果**

在使用 Tab 标签页时，可以将不同分页的界面布局保存在不同的 XML 文件中，也可以将所有分页的布局保存在同一个 XML 文件中。前一种方法有利于在 Eclipse 开发环境中进行可视化设计，并且不同分页的界面布局在不同的文件中更加易于管理，而后

一种方法则可以产生较少的 XML 文件，同时编码时的代码也会更加简洁。

## 5.3 界面布局

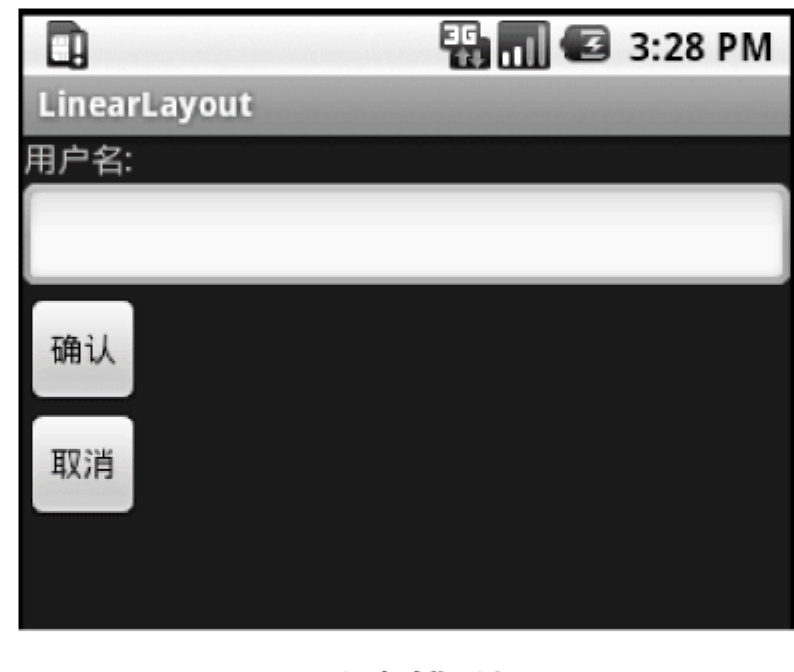
界面布局(Layout)是用户界面结构的描述，定义了界面中所有的元素、结构和相互关系。一般声明 Android 程序的界面布局有两种方法，第一种是使用 XML 文件描述界面布局，另一种是在程序运行时动态添加或修改界面布局。

Android 系统提供了在声明界面布局上提供了很好的灵活性，用户既可以独立使用任何一种声明界面布局的方式，也可以同时使用两种方式。一般情况下，XML 文件用来描述用户界面中最基本的元素，而在代码中动态修改需要更新状态的界面元素。当然，用户也可以在将所有的界面元素，无论在程序运行后是否需要修改其内容，都放在代码中进行定义和声明。很明显这不是一种好的设计方式，会给后期界面修改带来不必要的麻烦，而且界面元素较多时，程序的代码也会显得混乱不堪。

使用 XML 文件声明界面布局，能够更好地将程序的表现层和控制层分离，在后期修改用户界面时，无需更改程序的源代码。例如，在程序开发完成后，为了让程序能够支持不同屏幕尺寸、规格和语言的手机，则可以声明多个 XML 布局，而无需修改程序代码。不仅如此，使用 XML 文件声明的界面布局，用户还能够通过可视化工具直接看到所设计的用户界面，有利于加快界面设计的过程，并且为界面设计与开发带来极大的便利性。

### 5.3.1 线性布局

线性布局(LinearLayout)是一种重要的界面布局，也是经常使用到的一种界面布局。在线性布局中，所有的子元素都按照垂直或水平的顺序在界面上排列。如果垂直排列，则每行仅包含一个界面元素；同样，如果水平排列，则每列仅包含一个界面元素。图 5.12 分别是垂直排列的线性布局和水平排列的线性布局的示例。




(a) 垂直排列




(b) 水平排列

**图 5.12 线性布局**

下面将用一个简单的示例说明如何使用线性布局，示例的目标是实现图 5.13 所显示的用户界面，并对示例中用到的界面控件进行简单的介绍。

首先创建 Android 工程，工程名称是 LinearLayout，包名称是 edu. hrbeu.

LinearLayout，Activity 名称为 LinearLayout。为了能够完整体验创建线性布局的过程，首先删除 Eclipse 自动建立的/res/layout/main.xml 文件，然后建立用于显示垂直排列线性布局的 XML 文件。右击/res/layout 文件夹，选择 New→File 打开新文件建立向导，文件名为 main_vertical.xml，保存位置为 LinearLayout/res/layout，如图 5.13 所示。

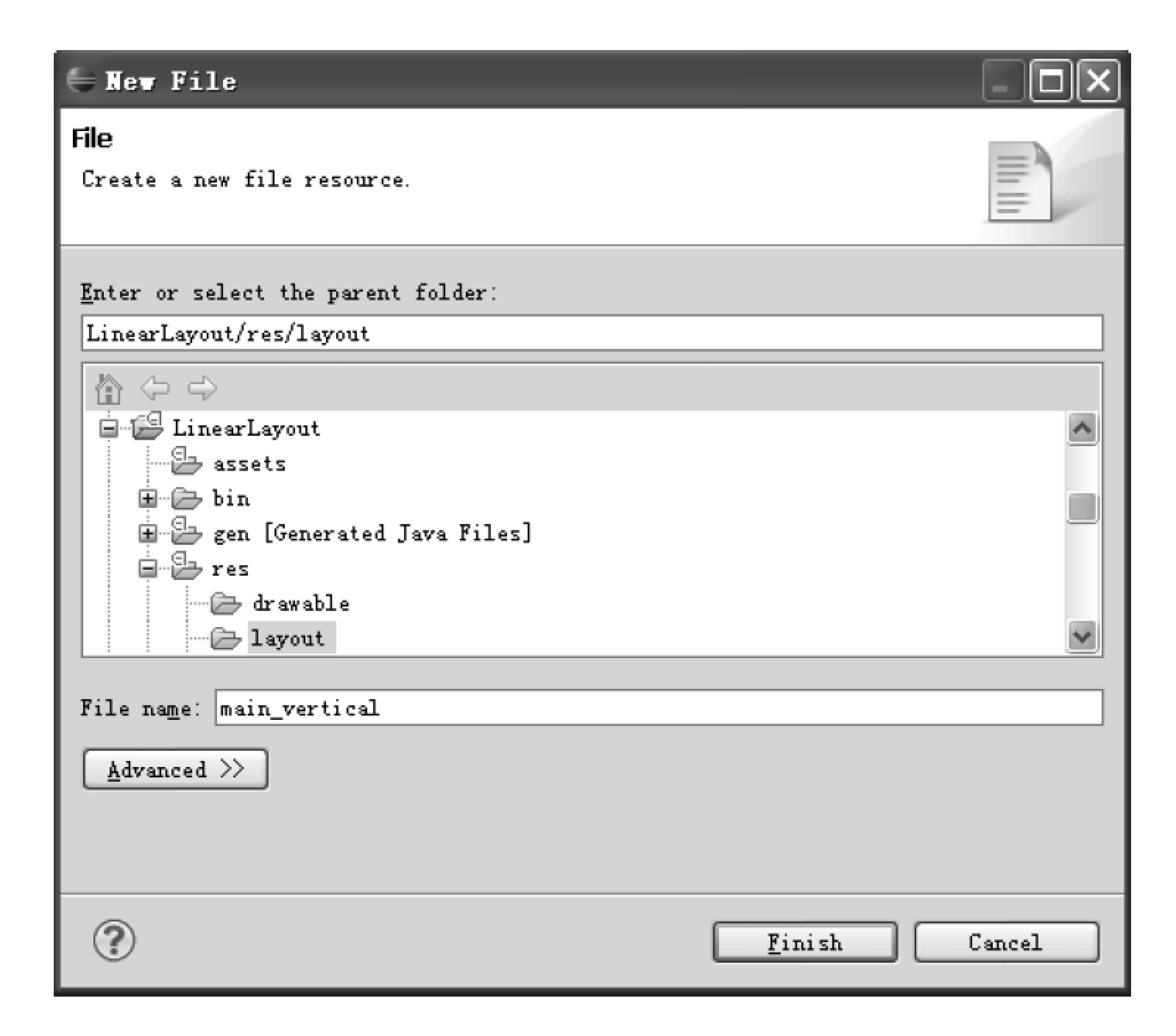



图 5.13　新文件建立向导

双击新建立的/res/layout/main_vertical.xml 文件，Eclipse 将打开界面布局的可视化编辑器，如图 5.14 所示。可视化编辑器顶部是资源配置清单，可以根据手机的配置不同选择不同的资源，主要用来实现应用软件的本地化。下部左侧是界面布局和界面控件，用户可以将需要的布局和控件拖拽到右面的可视化界面中，并修改布局和控件的属

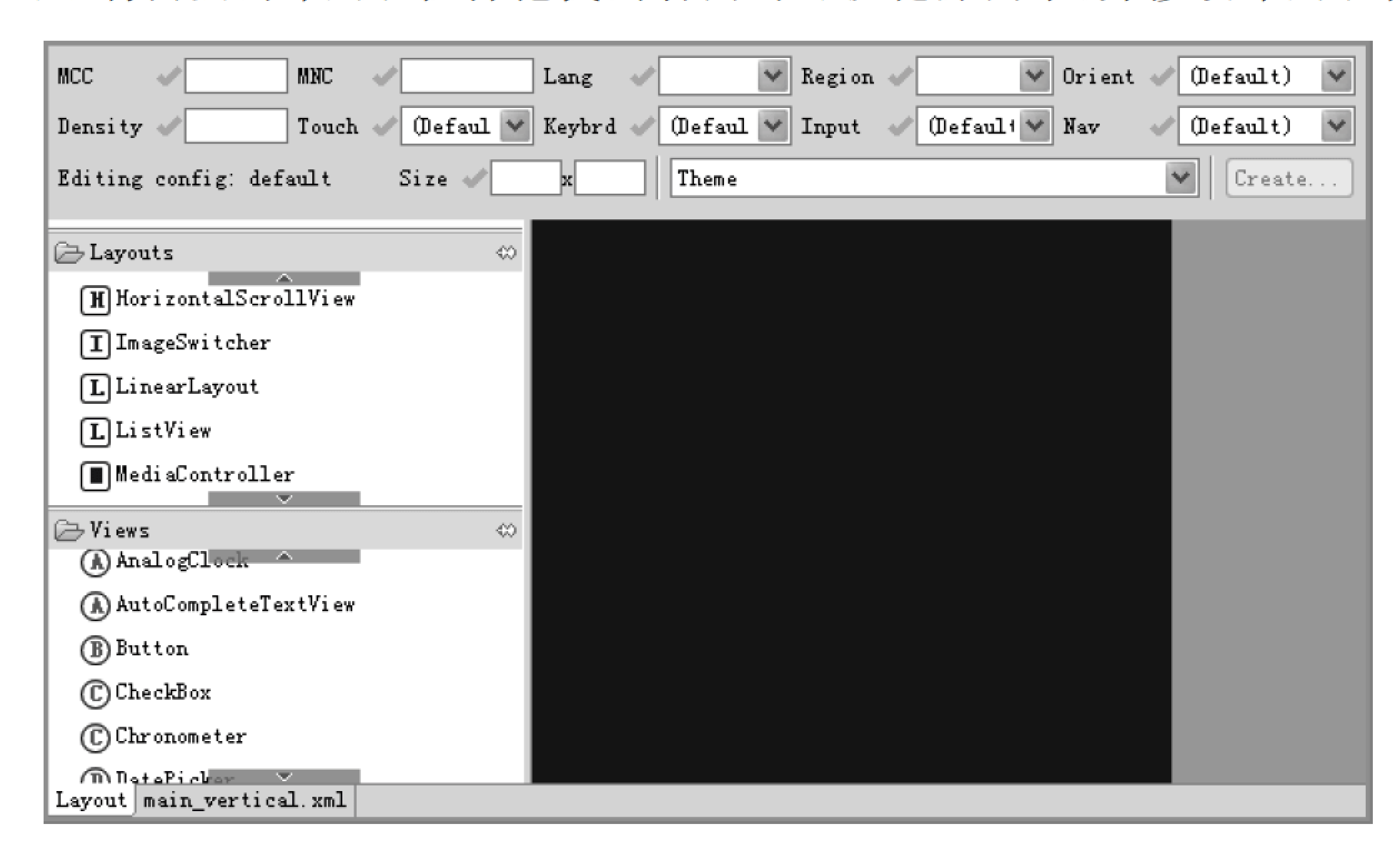



图 5.14　界面布局的可视化编辑器

性。右侧是可视化的用户界面，能够实时地呈现用户界面，但对无法正确显示中文。左下角的“Layout”和“main_vertical.xml”能够在可视化编辑器和 XML 文件编辑器之间切换。

在 Eclipse 右边的 Outline 中，双击 LinearLayout，打开线性布局的属性编辑器。线性布局的排列方法主要由 Orientation 属性进行控制，vertical 表示垂直排列，horizontal 表示水平排列。选择 Orientation 的值为 vertical，如图 5.15 所示，表示该线性布局为垂直排列。默认情况下，Layout height 的值为 wrap_content，表示线性布局高度等于所有子控件的高度总和，也就是线性布局的高度会刚好将所有子控件包含其中。将 Layout width 属性的值改为 fill_parent，表示线性布局宽度等于父控件的宽度，就是将线性布局在横向上占据父控件的所有空间。

| Property | Value |
|---|---|
| Next focus left | |
| Next focus right | |
| Next focus up | |
| Orientation | vertical<br>horizontal<br>vertical |
| Padding | |
| Padding bottom | |
| Padding left | |
| Padding right | |

**图 5.15　修改线性布局的 Orientation 属性**

打开 XML 文件编辑器，main_vertical.xml 文件的代码如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout
3       xmlns:android="http://schemas.android.com/apk/res/android"
4       android:layout_width="fill_parent"
5       android:layout_height="wrap_content"
6       android:orientation="vertical">
7   </LinearLayout>
```

第 2 行代码是声明 XML 文件的根元素为线性布局，第 4,5,6 行代码是在属性编辑器中修改过的宽度、高度和排列方式的属性。可见，用户在可视化编辑器和属性编辑器中的任何修改，都会同步反映在 XML 文件中；反之，用户对 XML 文件的修改，也会影响可视化编辑器和属性编辑器的内容。




**图 5.16　修改线性布局的 Orientation 属性**

然后，用户按照 TextView、EditText、Button、Button 的顺序，将 4 个界面控件先后拖拽到可视化编辑器中，所有控件会按照拖拽的顺序显示在可视化编辑器中，如图 5.16 所示。所有控件都自动获取控件名称，并把该名称显示在控件上，如 TextView01、EditText01、Button01 和 Button02 等。

将界面控件位置确定后，按照表 5.1 修改界面控件的属性。从表 5.1 中可以发

现，所有界面控件都有一个共同的属性 ID。ID 是一个字符串，编译时被转换为整数，可以用来在代码中引用界面元素，一般仅在代码中需要动态修改的界面元素时才为界面元素设置 ID，反之则不需要设置 ID。本例中没有代码中引用任何界面元素，因此完全可以不必设置 ID，但为了说明 ID 的用途和使用方法，在本例中为所有的界面控件设置 ID。

**表 5.1　线性布局界面控件的属性设置**

| 编号 | 类　型 | 属　　性 | 值 |
|---|---|---|---|
| 1 | TextView | Id | @+id/label |
| | | Text | 用户名： |
| 2 | EditText | Id | @+id/entry |
| | | Layout width | fill_parent |
| | | Text | [null] |
| 3 | Button | Id | @+id/ok |
| | | Text | 确认 |
| 4 | Button | Id | @+id/cancel |
| | | Text | 取消 |

在修改完所有界面控件的属性后，可以从可视化编辑器中发现，界面控件的中文字符都显示为“□”，这是因为可视化编辑器还不能很好地支持中文字符。但在打开 XML 文件编辑器查看 main_vertical.xml 文件代码，发现在属性编辑器内填入的文字已经正常写入到 XML 文件中，例如第 11，20，25 行代码。

```
1   <?xml version="1.0"encoding="utf-8"?>
2   <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="fill_parent"
5     android:layout_height="wrap_content"
6     android:orientation="vertical">
7
8       <TextView android:id="@+id/label"
9           android:layout_width="wrap_content"
10          android:layout_height="wrap_content"
11          android:text="用户名:">
12      </TextView>
13      <EditText android:id="@+id/entry"
14          android:layout_height="wrap_content"
15          android:layout_width="fill_parent">
16      </EditText>
```

```
        <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="确认">
        </Button>
        <Button android:id="@+id/cancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="取消">
        </Button>
</LinearLayout>
```

最后，将 LinearLayout.java 文件中的 setContentView(R.layout.main)，更改为 setContentView(R.layout.main_vertical)。运行后的结果如图 5.12(a)所示。

建立横向排列的线性布局的过程与上述的纵向线性布局非常相似，注意以下几个关键点即可：

(1) 建立 main_ horizontal.xml 文件。

(2) 线性布局的 Orientation 属性的值设置为 horizontal。

(3) 将 EditText 的 Layout width 属性的值设置为 wrap_content。

(4) 将 LinearLayout.java 文件中的 setContentView(R.layout.main_vertical) 修改为 setContentView(R.layout.main_ horizontal)。

### 5.3.2 框架布局

框架布局(FrameLayout)是最简单的界面布局，是用来存放一个元素的空白空间，且子元素的位置是不能够指定的，只能够放置在空白空间的左上角。如果有多个子元素，后放置的子元素将遮挡先放置的子元素。

为了更好地理解框架布局，使用 Android SDK 中提供的层级观察器(Hierarchy Viewer)进一步分析界面布局。层级观察器能够对用户界面进行分析和调试，并以图形化的方式展示树形结构的界面布局，另外，还提供了一个精确的像素级观察器(Pixel Perfect View)，以栅格的方式详细观察放大后的界面界面。

在模拟器上运行 5.3.1 节的垂直排列线性布局的示例，在层级观察器中获得示例界面布局的树形结构图，如图 5.17 所示。

结合界面布局的树形结构图(见图 5.17)和示意图(见图 5.18)，分析不同界面布局和界面控件的区域边界。用户界面的根节点(#0@43599ee0)是线性布局，其边界是整个界面，也就是示意图的最外层的实心线。根节点右侧的子节点(#0@43599a730)是框架布局，仅有一个节点元素(#0@4359ad18)，这个子元素是 TextView 控件，用来显示 Android 应用程序名称，其边界是示意图中的区域 1。因此框架布局元素 #0@43599a730 的边界是同区域 1 的高度相同，宽带充满整个根节点的区域。这两个界面元

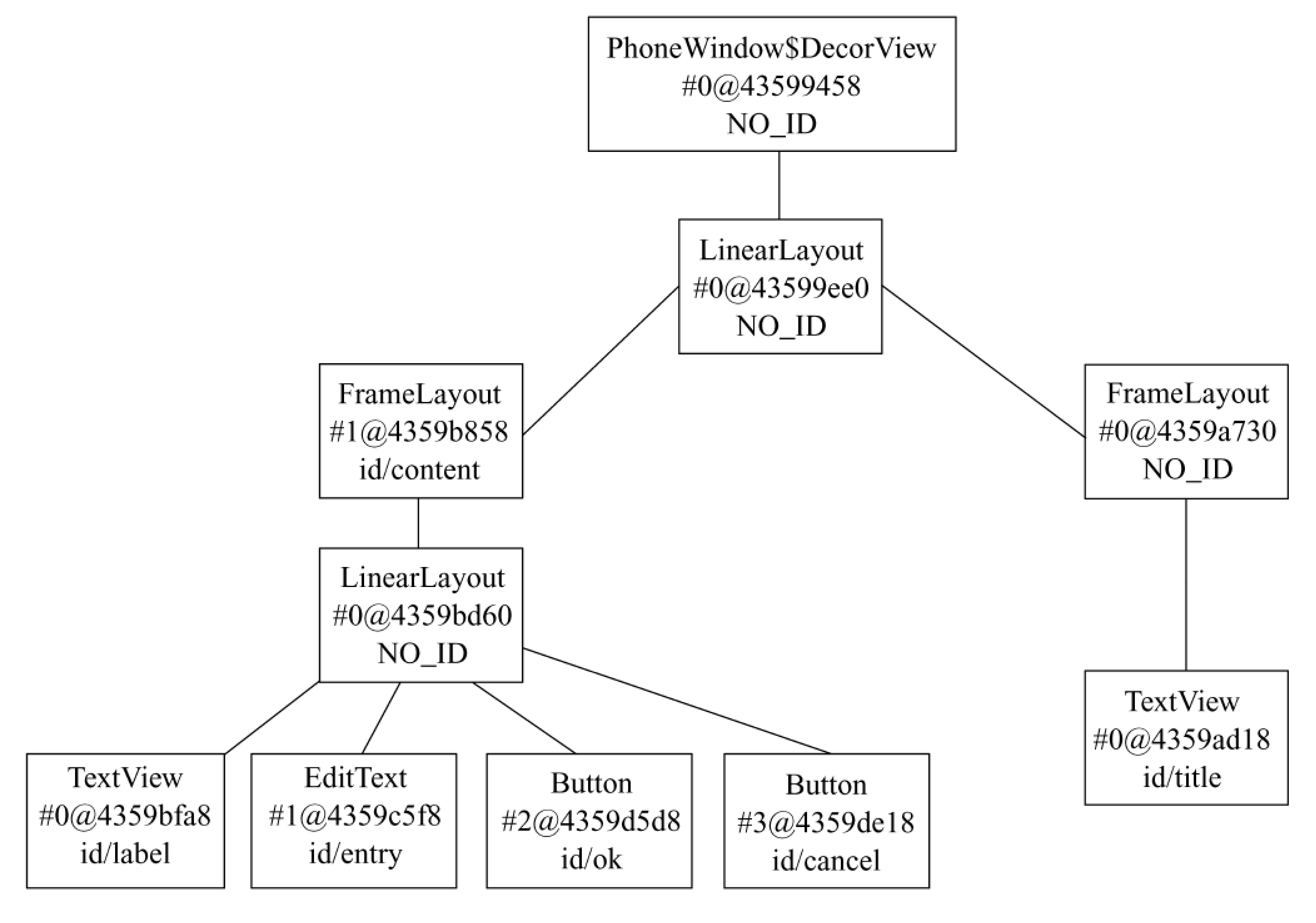



**图 5.17 界面布局的树形结构图**

素是系统自动生成的，一般情况下用户不能修改和编辑。

根节点左侧的子节点(#1@4359b858)也是框架布局，边界是区域2到区域7的全部空间。其下仅有一个子节点(#0@4359bd60)元素是线性布局，因为线性布局的Layout width属性设置为fill_parent，Layout height属性设置为wrap_content，因此该线性布局的宽度就是其父节点#1@4359b858的宽带，高度等于所有子节点元素的高度之和。线性布局#0@4359bd60的4个子节点元素#0@4359bfa8、#1@4359c5f8、#2@4359d5d8和#3@4359de18的边界，分别是界面布局示意图中的区域2、区域3、区域4和区域5。

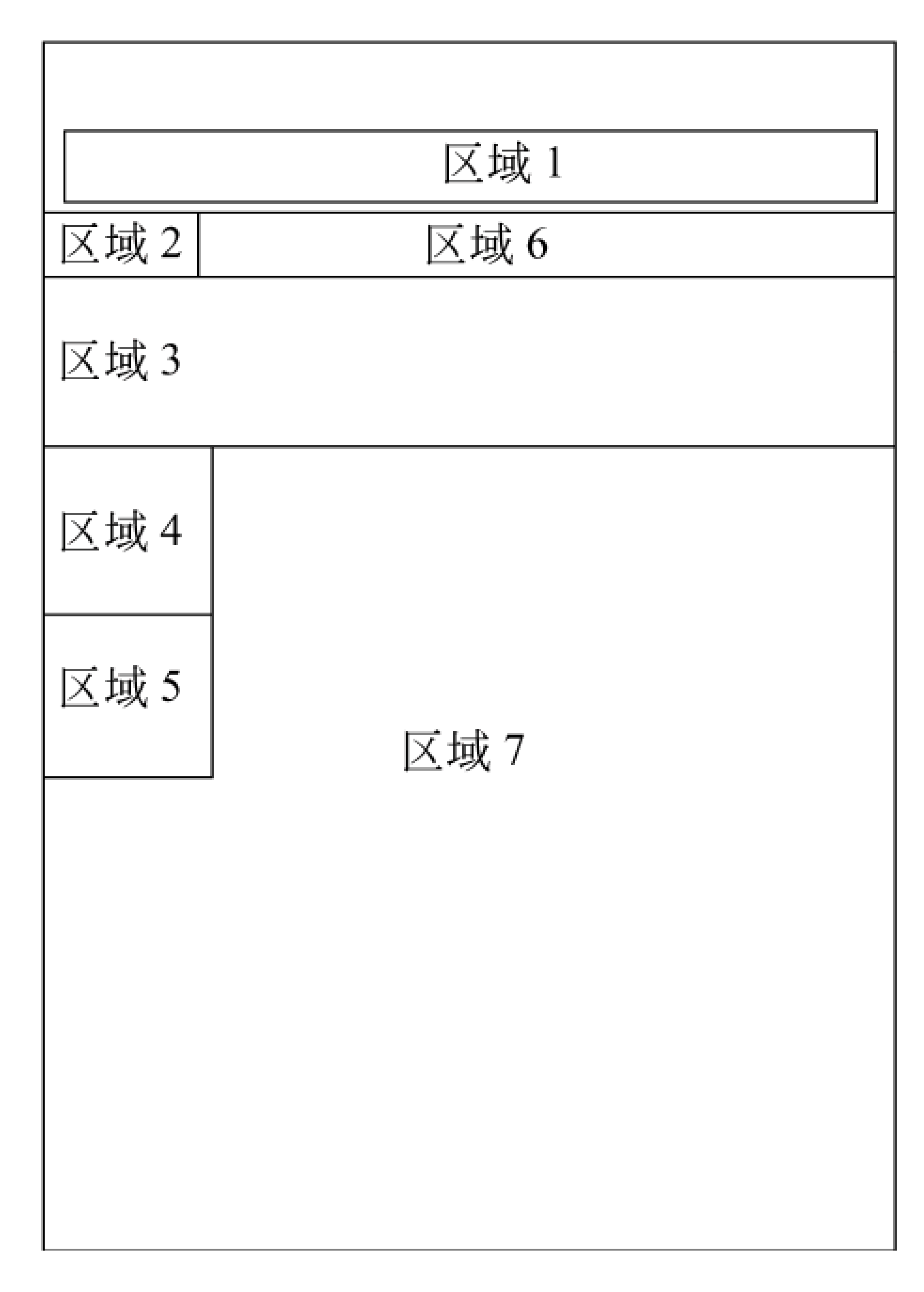



**图 5.18 界面布局的示意图**

## 5.3.3 表格布局

表格布局(TableLayout)也是一种常用的界面布局，它将屏幕划分网格，通过指定行和列可以将界面元素添加的网格中。对比网格布局的示意图(见图5.19)和效果图(见图5.20)，可以发现网格的边界对用户是不可见的。表格布局还支持嵌套，可以将另一个

表格布局放置在前一个表格布局的网格中，也可以在表格布局中添加其他界面布局，例如线性布局、相对布局等等。

参照5.3.2节所用的界面示例，使用表格布局实现用户界面。建立一个新的Android工程，工程名称为TableLayout，在/res/layout/main.xml文件中设计基于表格布局的用户界面。在表格布局中设计一个2×2的网格，每个网格中置放一个界面控件，实现效果如图5.20所示。

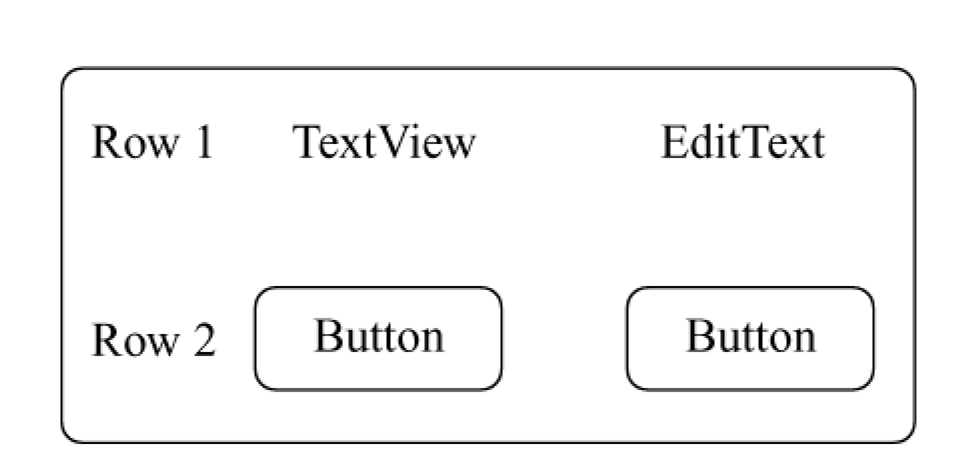



图5.19 表格布局的示意图




图5.20 表格布局的效果图

建立表格布局的示例并不困难，参照5.3.2节的示例可以很容易地实现，但要注意以下几点：

(1) 向界面中添加一个线性布局，无需修改布局的属性值。其中，Id属性为TableLayout01，Layout width和Layout height属性都为wrap_content。

(2) 向TableLayout01中添加两个TableRow。TableRow代表一个单独的行，每行被划分为几个小的单元，单元中可以添加一个界面控件。其中，Id属性分别为TableRow01和TableRow02，Layout width和Layout height属性都为wrap_content。

(3) 通过Outline，向TableRow01中添加TextView和EditText，如图5.21所示。

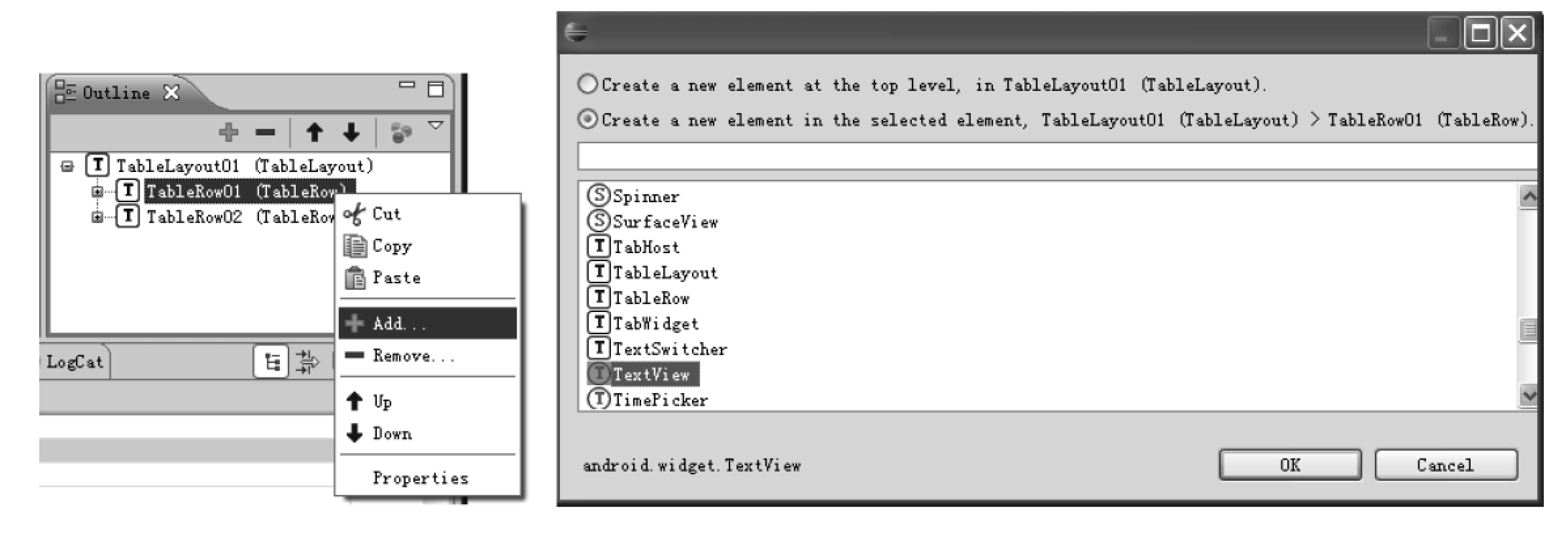



图5.21 向TableRow中添加界面控件

(4) 通过Outline向TableRow02中添加两个Button。

(5) 参考表5.2设置TableRow中4个界面控件的属性值。

表 5.2 表格布局界面控件的属性设置

| 编号 | 类 型 | 属 性 | 值 |
|---|---|---|---|
| 1 | TextView | Id | @+id/label |
| | | Text | 用户名： |
| | | Gravity | right |
| | | Padding | 3dip |
| | | Layout width | 160dip |
| 2 | EditText | Id | @+id/entry |
| | | Text | [null] |
| | | Padding | 3dip |
| | | Layout width | 160dip |
| 3 | Button | Id | @+id/ok |
| | | Text | 确认 |
| | | Padding | 3dip |
| 4 | Button | Id | @+id/cancel |
| | | Text | 取消 |
| | | Padding | 3dip |

main.xml 文件的完整代码如下：

```
<?xml version="1.0"encoding="utf-8"?>

<TableLayout android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TableRow android:id="@+id/TableRow01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView android:id="@+id/label"
            android:layout_height="wrap_content"
            android:layout_width="160dip"
            android:gravity="right"
            android:text="用户名："
            android:padding="3dip">
        </TextView>
        <EditText android:id="@+id/entry"
            android:layout_height="wrap_content"
            android:layout_width="160dip"
```

```
20                  android:padding="3dip">
21              </EditText>
22          </TableRow>
23          <TableRow android:id="@+id/TableRow02"
24              android:layout_width="wrap_content"
25              android:layout_height="wrap_content">
26              <Button android:id="@+id/ok"
27                  android:layout_height="wrap_content"
28                  android:padding="3dip"
29                  android:text="确认">
30              </Button>
31              <Button android:id="@+id/Button02"
32                  android:layout_width="wrap_content"
33                  android:layout_height="wrap_content"
34                  android:padding="3dip"
35                  android:text="取消">
36              </Button>
37          </TableRow>
38      </TableLayout>
```

第 3 行代码使用了＜TableLayout＞标签声明表格布局；第 7 行和第 23 行代码声明了两个 TableRow 元素，用来表示布局中的两行；第 12 行利用宽度属性 android:layout_width，将 TextView 元素的宽度指定为 160dip；第 13 行使用属性 android:gravity，将 TextView 中的文字对齐方式指定为右对齐；第 15 行使用属性 android:padding，声明 TextView 元素与其他元素的间隔距离为 3dip。

## 5.3.4 相对布局

相对布局(RelativeLayout)是一种非常灵活的布局方式，能够通过指定界面元素与其他元素的相对位置关系，确定界面中所有元素的布局位置。相对布局和线性布局有着共同的优点，能够最大程度保证在各种屏幕类型的手机上正确显示界面布局。

图 5.22 是相对布局的一个示例，为了便于理解，先用文字描述对界面元素的添加顺序和相互关系进行说明。添加 TextView 控件(“用户名”)，相对布局会将 TextView 控件放置在屏幕的最上方；然后添加 EditText 控件(输入框)，并声明该控件的位置在 TextView 控件的下方，相对布局会根据 TextView 的位置确定 EditText 控件的位置；之后添加第一个 Button 控件(“取消”按钮)，声明在 EditText 控件的下方，且在父控件的最右边；最后，添加第二个 Button 控件(“确认”按钮)，声明该控件在第一个 Button 控件的左方，且与第一个 Button 控件处于相同的水平位置。

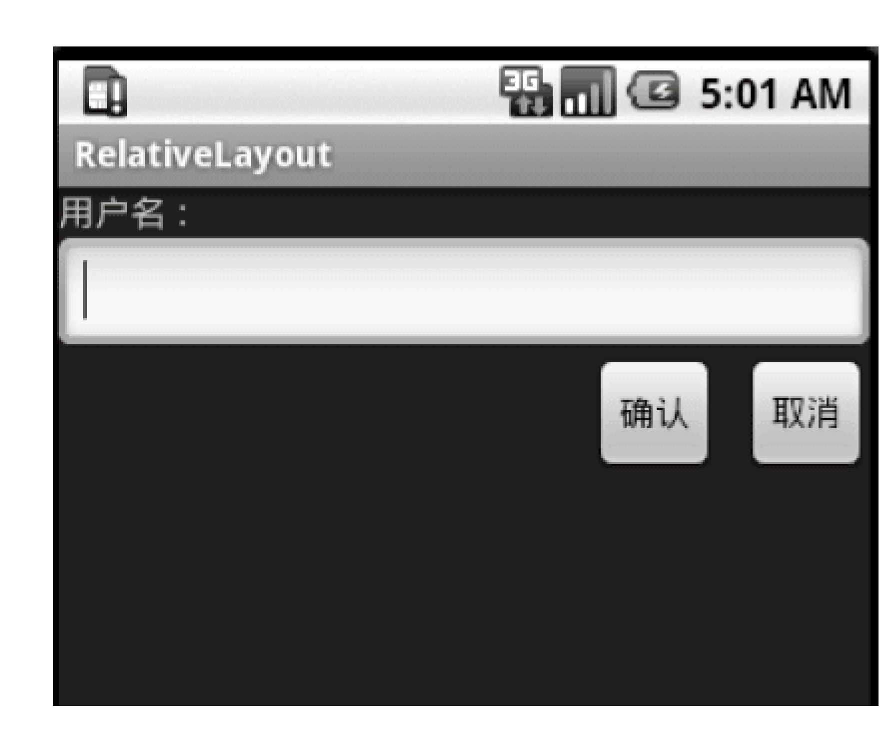



图 5.22 相对布局

main.xml 文件的完整代码如下：

```
1  <?xml version="1.0"encoding="utf-8"?>
2
3  <RelativeLayout android:id="@+id/RelativeLayout01"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      xmlns:android="http://schemas.android.com/apk/res/android">
7      <TextView android:id="@+id/label"
8          android:layout_height="wrap_content"
9          android:layout_width="fill_parent"
10         android:text="用户名：">
11     </TextView>
12     <EditText android:id="@+id/entry"
13         android:layout_height="wrap_content"
14         android:layout_width="fill_parent"
15         android:layout_below="@id/label">
16     </EditText>
17     <Button android:id="@+id/cancel"
18         android:layout_height="wrap_content"
19         android:layout_width="wrap_content"
20         android:layout_alignParentRight="true"
21         android:layout_marginLeft="10dip"
22         android:layout_below="@id/entry"
23         android:text="取消">
24     </Button>
25         <Button android:id="@+id/ok"
26         android:layout_height="wrap_content"
27         android:layout_width="wrap_content"
28         android:layout_toLeftOf="@id/cancel"
29         android:layout_alignTop="@id/cancel"
30         android:text="确认">
31     </Button>
32 </RelativeLayout>
```

在上面的代码中，首先在第 3 行使用了<RelativeLayout>标签声明一个相对布局；第 15 行使用位置属性 android:layout_below，确定 EditText 控件在 ID 为 label 的元素下方；第 20 行使用属性 android:layout_alignParentRight，声明该元素与其父元素的右边边界对齐；第 21 行使用属性 android:layout_marginLeft，将该元素向左移动 10dip；第 22 行声明该元素在 ID 为 entry 的元素下方；第 28 行声明使用属性 android:layout_toLeftOf，声明该元素在 ID 为 cancel 元素的左边；第 29 行使用属性 android:layout_alignTop，声明该元素与 ID 为 cancel 的元素在相同的水平位置。

### 5.3.5 绝对布局

绝对布局(AbsoluteLayout)能通过指定界面元素的坐标位置，来确定用户界面的整

体布局。绝对布局是一种不推荐使用的界面布局,因为通过X轴和Y轴确定界面元素位置后,Android系统不能够根据不同屏幕对界面元素的位置进行调整,降低了界面布局对不同类型和尺寸屏幕的适应能力。使用绝对布局往往在目标手机上非常完美,但在其他不同类型的手机上,界面布局却变得混乱不堪。




图5.23　绝对布局

图5.23是绝对布局的一个示例,每一个界面控件都必须指定坐标(X,Y),例如"确认"按钮的坐标是(40,120),"取消"按钮的坐标是(120,120)。坐标原点(0,0)在屏幕的左上角。

下面给出main.xml文件的完整代码:

```
<?xml version="1.0" encoding="utf-8"?>

<AbsoluteLayout android:id="@+id/AbsoluteLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TextView android:id="@+id/label"
        android:layout_x="40dip"
        android:layout_y="40dip"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="用户名：">
    </TextView>
    <EditText android:id="@+id/entry"
        android:layout_x="40dip"
        android:layout_y="60dip"
        android:layout_height="wrap_content"
        android:layout_width="150dip">
    </EditText>
    <Button android:id="@+id/ok"
        android:layout_width="70dip"
        android:layout_height="wrap_content"
        android:layout_x="40dip"
        android:layout_y="120dip"
        android:text="确认">
    </Button>
    <Button android:id="@+id/cancel"
        android:layout_width="70dip"
        android:layout_height="wrap_content"
        android:layout_x="120dip"
        android:layout_y="120dip"
        android:text="取消">
```

```
        </Button>
    </AbsoluteLayout>
```

# 5.4 菜　　单

菜单是应用程序中非常重要的组成部分，能够在不占用界面空间的前提下，为应用程序提供了统一的功能和设置界面，并为程序开发人员提供了易于使用的编程接口。Android 系统支持三种菜单，即选项菜单（Option Menu）、子菜单（Submenu）和快捷菜单（Context Menu）。

## 5.4.1 选项菜单

选项菜单是一种经常被使用的 Android 系统菜单，用户可以通过“菜单键”（MENU key）打开选项菜单。选项菜单分为两种，即图标菜单（Icon Menu）和扩展菜单（Expanded Menu），通过“菜单键”直接打开的是图标菜单，如图 5.24 所示。顾名思义，图标菜单就是能够同时显示文字和图标的菜单，最多支持 6 个子项，如果子项多余 6 个，则需要扩展菜单显示其他的子项。

扩展菜单是垂直的列表型菜单，如图 5.25 所示，仅在图标菜单子项多于 6 个时才出现，通过点击图标菜单最后的子项 More 才能打开。扩展菜单不能显示图标，但支持单选框和复选框；相反，图标菜单支持显示图标，但不支持单选框和复选框。

菜单子项0 菜单子项1 菜单子项2
菜单子项3 菜单子项4 More

图 5.24　图标菜单




图 5.25　扩展菜单

为了能够在 Android 应用程序中使用选项菜单，必须重载 Activity 的 onCreateOptionsMenu()函数。该函数在用户首次使用选项菜单时被调用，一般用来初始化菜单子项的相关内容，包括设置菜单子项自身的子项 ID 和组 ID，菜单子项显示的文字和图片等信息。

```
final static int MENU_DOWNLOAD=Menu.FIRST;
final static int MENU_UPLOAD=Menu.FIRST+1;
@Override
public boolean onCreateOptionsMenu(Menu menu){
```

```
5            menu.add(0,MENU_DOWNLOAD,0,"下载设置");
6            menu.add(0,MENU_UPLOAD,1,"上传设置");
7            return true;
8       }
```

一般将菜单子项 ID 定义成静态常量(第 1 行和第 2 行),并使用静态常量 Menu.FIRST(整数类型,值为 1)定义第一个菜单子项,以后的菜单子项仅需在 Menu.FIRST 增加相应的数值即可。在 onCreateOptionsMenu()函数中,函数的返回值类型为布尔(第 7 行),返回 true 将显示在函数中设置的菜单,否则将不能够显示菜单。Menu 对象(第 4 行)作为一个参数被传递到函数内部,因此在 onCreateOptionsMenu()函数中,用户可以使用 Menu 对象的 add()函数添加菜单子项。add()函数的语法如下:

```
MenuItem android.view.Menu.add(int groupId, int itemId, int order, CharSequence title)
```

add()函数的第 1 个参数 groupId 是组 ID,用以批量地对菜单子项进行处理和排序;第 2 关参数 itemId 是子项 ID,是每一个菜单子项的唯一标识,通过子项 ID 使应用程序能够定位到用户所选择的菜单子项;第 3 个参数 order 是定义菜单子项在选项菜单中的排列顺序;第 4 个参数 title 是菜单子项所显示的标题。

另外,还可以通过 setIcon()函数和 setShortcut()函数,为菜单子项添加图标和快捷键。setShortcut()函数第一个参数是为数字键盘设定的快捷键,第二个参数是为全键盘设定的快捷键,且不区分字母的大小写。MENU_DOWNLOAD 菜单设置图标和快捷键的代码。

```
1       menu.add(0,MENU_DOWNLOAD,0,"下载设置")
2           .setIcon(R.drawable.download);
3           .setShortcut('0','d');
```

第 2 行代码中使用了新的图像资源,用户将需要使用的图像文件拷贝到/res/drawable 目录下。

onCreateOptionsMenu()函数仅在第一次使用菜单时被调用一次,如果希望打开菜单时,能够动态的添加、删除菜单子项,或修改菜单的标题、图标和可见性等内容,则需要重载 onPrepareOptionsMenu()函数。onPrepareOptionsMenu()函数的返回值的含义与 onCreateOptionsMenu()函数相同,返回 true 则显示菜单,返回 false 则不显示菜单。下面的代码是在用户每次打开选项菜单时,在菜单子项中显示用户打开该子项的次数。

```
1       static int MenuUploadCounter=0;
2       @Override
3       public boolean onPrepareOptionsMenu(Menu menu){
4           MenuItem uploadItem=menu.findItem(MENU_UPLOAD);
5           uploadItem.setTitle("上传设置:"+
                String.valueOf(MenuUploadCounter));
6           return true;
7       }
```

代码中设置一个菜单子项的计数器(第 1 行),用来统计用户打开“上传设置”子项的

次数。通过将菜单子项的 ID 传递给 menu.findItem()函数，获取到菜单子项的对象(第 4 行)，并且通过 MenuItem 的 setTitle()函数修改菜单的标题(第 5 行)。

在 Android 系统中，所有菜单子项的都使用同一个处理函数 onOptionsItemSelected()，对菜单选择事件进行处理，且该函数在每次用户单击菜单子项时都会被调用。下面的代码说明了如何通过菜单子项的子项 ID 执行不同的操作。

```
1   @Override
2   public boolean onOptionsItemSelected(MenuItem item){
3       switch(item.getItemId()){
4           case MENU_DOWNLOAD:
5               MenuDownlaodCounter++;
6               return true;
7           case MENU_UPLOAD:
8               MenuUploadCounter++;
9               return true;
10      }
11      return false;
12  }
```

onOptionsItemSelected()的返回值表示是否对菜单的选择事件进行处理，如果已经处理过则返回 true，否则返回 false。第 2 行的 MenuItem.getItemId()函数可以获取到被选择菜单子项的 ID。

完整代码请参考 OptionsMenu 程序，程序运行后，通过单击“菜单键”可以调出程序设计的两个菜单子项，如图 5.26 所示。




图 5.26 OptionsMenu 程序运行结果




图 5.27 子菜单

## 5.4.2 子菜单

子菜单是能够显示更加详细信息的菜单子项。传统的子菜单一般采用树形层次化

结构，但 Android 系统却使用了浮动窗体的形式显示菜单子项，如图 5.27 所示。采用与众不同的显示方式，主要是为了更好适应小屏幕的显示方式。

Android 系统的子菜单使用非常灵活，可以在选项菜单或快捷菜单中使用子菜单，有利于将相同或相似的菜单子项组织在一起，便于显示和分类。但是 Android 系统的子菜单不支持嵌套，也就是说不能够在子菜单中再使用子菜单。

子菜单的添加是通过 addSubMenu()函数实现的，例如在下面的第 1 行代码中，在 onCreateOptionsMenu()函数传递的 menu 对象上调用 addSubMenu()函数，在选项菜单中添加一个菜单子项，用户点击后可以打开子菜单。addSubMenu()函数与选项菜单中使用过的 add()函数支持相同的参数，同样可以指定菜单子项的 ID、组 ID 和标题等参数，并且是能够通过 setIcon()函数菜单显示的图标。

```
1    SubMenu uploadMenu= (SubMenu) menu.addSubMenu(0,MENU_UPLOAD,1,"上传设置")
                                                  .setIcon(R.drawable.upload);
2    uploadMenu.setHeaderIcon(R.drawable.upload);
3    uploadMenu.setHeaderTitle("上传参数设置");
4    uploadMenu.add(0,SUB_MENU_UPLOAD_A,0,"上传参数 A");
5    uploadMenu.add(0,SUB_MENU_UPLOAD_B,0,"上传参数 B");
```

代码第 2 行使用 setHeaderIcon()函数，定义了子菜单的图标；第 3 行定义了子菜单的标题，如果不规定子菜单的标题，子菜单将显示父菜单子项的标题，即第 1 行代码中的“上传设置”；第 4 行和第 5 行在子菜单中添加了两个菜单子项，菜单子项的更新函数和选择事件处理函数，仍然使用 onPrepareOptionsMenu()函数和 onOptionsItemSelected()函数。

以 5.4.1 节的代码为基础，将“上传设置”改为子菜单，并在子菜单中添加“上传参数 A”和“上传参数 B”两个菜单子项。完整代码请参考 MySubMenu 程序，运行结果如图 5.28 所示。




**图 5.28 MySubMenu 程序的运行结果**

## 5.4.3 快捷菜单

快捷菜单同样采用了动窗体的显示方式，虽然快捷菜单的实现方式与子菜单相同，但两种菜单的启动方式却截然不同。快捷菜单类似于普通桌面程序中的“右键菜单”，当用户单击界面元素超过 2 秒后，将启动注册到该界面元素的快捷菜单。

使用快捷菜单的方法与使用选项菜单的方法非常相似，需要重载 onCreateContextMenu()函数和 onContextItemSelected()函数。onCreateContextMenu()函数主要用来添加快捷菜单所显示的标题、图标和菜单子项等内容，选项菜单中的 onCreateOptionsMenu()函数仅在选项菜单第一次启动时被调用一次，但快捷菜单的 onCreateContextMenu()函数每次启动时都会被调用一次。

```
1    final static int CONTEXT_MENU_1=Menu.FIRST;
2    final static int CONTEXT_MENU_2=Menu.FIRST+1;
3    final static int CONTEXT_MENU_3=Menu.FIRST+2;
4    @Override
5    public void onCreateContextMenu(ContextMenu menu, View v,
             ContextMenuInfo menuInfo){
6        menu.setHeaderTitle("快捷菜单标题");
7        menu.add(0, CONTEXT_MENU_1, 0,"菜单子项 1");
8        menu.add(0, CONTEXT_MENU_2, 1,"菜单子项 2");
9        menu.add(0, CONTEXT_MENU_3, 2,"菜单子项 3");
10   }
```

ContextMenu 类支持 add()函数(代码第 7 行)和 addSubMenu()函数,可以在快捷菜单中添加菜单子项和子菜单。onCreateContextMenu()函数(代码第 5 行)的第 1 个参数 menu 是需要显示的快捷菜单;第 2 个参数 v 是用户选择的界面元素;第 3 个参数 menuInfo 是所选择界面元素的额外信息。

菜单选择事件的处理需要重载 onContextItemSelected()函数,该函数在用户选择快捷菜单中的菜单子项后被调用,与 onOptionsItemSelected ()函数的使用方法基本相同。

```
1        @Override
2        public boolean onContextItemSelected(MenuItem item){
3            switch(item.getItemId()){
4                case CONTEXT_MENU_1:
5                    LabelView.setText("菜单子项 1");
6                    return true;
7                case CONTEXT_MENU_2:
8                    LabelView.setText("菜单子项 2");
9                    return true;
10               case CONTEXT_MENU_3:
11                   LabelView.setText("菜单子项 3");
12           return true;
13           }
14       return false;
15       }
```

最后,还需要使用 registerForContextMenu()函数,将快捷菜单注册到界面控件上(下方代码第 7 行)。这样,用户在长时间单击该界面控件时,便会启动快捷菜单。同时,为了能够在界面上直接显示用户所选择快捷菜单的菜单子项,在代码中引用了界面元素 TextView(下方代码第 6 行),通过更改 TextView 的显示内容(上方代码第 5,8 和 11 行),显示用户所选择的菜单子项。

```
1    TextView LabelView=null;
2    @Override
3    public void onCreate(Bundle savedInstanceState) {
```

```
4        super.onCreate(savedInstanceState);
5        setContentView(R.layout.main);
6        LabelView= (TextView)findViewById(R.id.label);
7        registerForContextMenu(LabelView);
8    }
```

下方代码是/src/layout/main. xml 文件的部分内容，第 1 行声明了 TextView 的 ID 为 label，在上方代码的第 6 行中，通过 R. id. label 将 ID 传递给 findViewById()函数，这样用户便能够引用该界面元素，并能够修改该界面元素的显示内容。

```
1    <TextView    android:id="@+id/label"
2        android:layout_width="fill_parent"
3        android:layout_height="fill_parent"
4        android:text="@string/hello"
5    />
```

还有一点需要注意，上方代码的第 2 行，将 android:layout_width 设置为 fill_parent，这样 TextView 将填充满父节点的所有剩余屏幕空间，用户单击屏幕 TextView 下方任何位置都可以启动快捷菜单。如果将 android:layout_width 设置为 wrap_content，则用户必须准确单击 TextView 才能启动快捷菜单。完整代码参考 MyContextMenu 程序，运行结果如图5.29所示。




**图 5.29　MyContextMen 程序的运行结果**

在 Android 系统中，菜单不仅能够在代码中定义，而且可以像界面布局一样在 XML 文件中进行定义。使用 XML 文件定义界面菜单，将代码与界面设计分类，有助于简化代码的复杂程度，并且更有利于界面的可视化。下面将快捷菜的示例程序 MyContextMen 改用 XML 实现，新程序的工程名称为 MyXLMContoxtMenu。

首先需要创建保存菜单内容的 XML 文件。在/src 目录下建立子目录 menu，并在 menu 下建立 context_menu. xml 文件，代码如下：

```
1    <menu xmlns:android="http://schemas.android.com/apk/res/android">
2        <item android:id="@+id/contextMenu1"
3            android:title="菜单子项 1"/>
4        <item android:id="@+id/contextMenu2"
5            android:title="菜单子项 2"/>
6        <item android:id="@+id/contextMenu3"
7            android:title="菜单子项 3"/>
8    </menu>
```

在描述菜单的 XML 文件中，必须以<menu>标签（代码第 1 行）作为根节点，<item>标签（代码第 2 行）用来描述菜单中的子项，<item>标签可以通过嵌套实现子菜单的功

能。XML 菜单的显示结果如图 5.30 所示。

在 XML 文件中定义菜单后，在 onCreateContextMenu()函数中调用 inflater.inflate()方法，将 XML 资源文件传递给菜单对象。

菜单子项1

菜单子项2

菜单子项3

图 5.30 XML 菜单

```
1     @Override
2     public void onCreateContextMenu(ContextMenu menu,
3                   View v, ContextMenuInfo menuInfo){
4         MenuInflater inflater=getMenuInflater();
5         inflater.inflate(R.menu.context_menu, menu);
6     }
```

在上述代码第 4 行，getMenuInflater()为当前的 Activity 返回 MenuInflater。然后，通过第 5 行代码，将 XML 资源文件 R.menu.context_menu，传递给 menu 这个快捷菜单对象。

# 5.5 界面事件

在 Android 系统中，存在多种界面事件，如按键事件、触摸事件、焦点事件和菜单事件等等，在这些事件发生时，Android 界面框架调用界面控件的事件处理函数对事件进行处理。

## 5.5.1 按键事件

在 MVC 模型中，控制器根据界面事件(UI Event)类型不同，将事件传递给界面控件不同的事件处理函数。例如按键事件(KeyEvent)将传递给 onKey()函数进行处理，触摸事件(TouchEvent)将传递给 onTouch()函数进行处理。

Android 系统界面事件的传递和处理遵循着一定的规则。首先，如果界面控件设置了事件监听器，则事件将先传递给事件监听器；相反，如果界面控件没有设置事件监听器，界面事件则会直接传递给界面控件的其他事件处理函数。即使界面控件设置了事件监听器，界面事件也可以再次传递给其他事件处理函数，是否继续传递事件给其他处理函数是由事件监听器处理函数的返回值决定的。如果监听器处理函数的返回值为 true，表示该事件已经完成处理过程，不需要其他处理函数参与处理过程，这样事件就不会再继续进行传递。反之，如果监听器处理函数的返回值为 false，则表示该事件没有完成处理过程，或需要其他处理函数捕获到该事件，事件会被传递给其他的事件处理函数。

以 EditText 控件中的按键事件为例，说明 Android 系统界面事件传递和处理过程，假设 EditText 控件已经设置了按键事件监听器。当用户按下键盘上的某个按键时，控制器将产生 KeyEvent 按键事件。Android 系统会首先判断 EditText 控件是否设置了按键事件监听器，因为 EditText 控件已经设置按键事件监听器 OnKeyListener，所以按键事

件先传递到监听器的事件处理函数 onKey()中。事件能够继续传递给 EditText 控件的其他事件处理函数,完全根据 onKey()函数的返回值来确定。如果 onKey()函数返回 false,事件将继续传递,这样 EditText 控件就可以捕获到该事件,将按键的内容显示在 EditText 控件中。如果 onKey()函数返回 true,将阻止按键事件的继续传递,这样 EditText 控件就不能捕获到按键事件,也就不能将按键内容显示在 EditText 控件中。

Android 界面框架支持对按键事件的监听,并能够将按键事件的详细信息传递给处理函数。为了处理控件的按键事件,先需要设置按键事件的监听器,并重载 onKey()函数。示例代码如下:

```
1    entryText.setOnKeyListener(new OnKeyListener(){
2        @Override
3        public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4            //过程代码…
5            return true/false;
6        }
```

第 1 行代码是设置控件的按键事件监听器。在代码第 3 行的 onKey()函数中,第 1 个参数 view 表示产生按键事件的界面控件;第 2 个参数 keyCode 表示按键代码;第 3 个参数 keyEvent 则包含了事件的详细信息,如按键的重复次数、硬件编码和按键标志等。第 5 行是 onKey()函数的返回值,返回 true,阻止事件传递;返回 false,允许继续传递按键事件。

KeyEventDemo 是一个说明如何处理按键事件的示例。在这个示例中,用户界面如图 5.31 所示。




**图 5.31　KeyEventDemo 界面**

在 KeyEventDemo 的用户界面中,最上方的 EditText 控件是输入字符的区域,中间的 CheckBox 控件用来控制 onKey()函数的返回值,最下方的 TextView 控件用来显示按键事件的详细信息,包括按键动作、按键代码、按键字符、Unicode 编码、重复次数、功能键状态、硬件编码和按键标志。界面的 XML 文件的代码如下:

```
<EditText android:id="@+id/entry"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</EditText>
<CheckBox android:id="@+id/block"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="返回 true,阻止将按键事件传递给界面元素">
</CheckBox>
<TextView android:id="@+id/label"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="按键事件信息">
</TextView>
```

在 EditText 中,每当任何一个键子按下或抬起时,都会引发按键事件。但为了能够使 EditText 处理按键事件,需要使用 setOnKeyListener()函数在代码中设置按键事件监听器,并在 onKey()函数添加按键事件的处理过程。

```
1  entryText.setOnKeyListener(new OnKeyListener(){
2      @Override
3      public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4          int metaState= keyEvent.getMetaState();
5          int unicodeChar= keyEvent.getUnicodeChar();
6          String msg= "";
7          msg +="按键动作:"+ String.valueOf(keyEvent.getAction())+ "\n";
8          msg +="按键代码:"+ String.valueOf(keyCode)+ "\n";
9          msg +="按键字符:"+ (char)unicodeChar+ "\n";
10         msg +="UNICODE:"+ String.valueOf(unicodeChar)+ "\n";
11         msg +="重复次数:"+ String.valueOf(keyEvent.getRepeatCount())+ "\n";
12         msg +="功能键状态:"+ String.valueOf(metaState)+ "\n";
13         msg +="硬件编码:"+ String.valueOf(keyEvent.getScanCode())+ "\n";
14         msg +="按键标志:"+ String.valueOf(keyEvent.getFlags())+ "\n";
15         labelView.setText(msg);
16         if (checkBox.isChecked())
17             return true;
18         else
19             return false;
20         }
```

第 4 行代码用来获取功能键状态。功能键包括左 Alt 键、右 Alt 键和 Shift 键,当这三个功能键被按下时,功能键代码 metaState 值分别为 18,34 和 65;但没有功能键被按下时,功能键代码 metaState 值分别为 0。第 5 行代码获取了按键的 Unicode 值,在第 9 行中,将 Unicode 转换为字符,显示在 TextView 中。第 7 行代码获取了按键动作,0 表示按下按键,1 表示抬起按键。第 7 行代码获取按键的重复次数,但按键被长时间按下时,则会产生这个属性值。第 13 行代码获取了按键的硬件编码,不同硬件设备的按键硬件编码都不相同,因此该值一般用于调试。第 14 行获取了按键事件的标志符。

### 5.5.2 触摸事件

触摸屏已经不是高端手机才有的配件,随着触摸屏生产成本的不断下降,目前很多中低端的手机也都装有触摸屏。伴随着触摸屏的普及,手机的操作方式也随之改变,用户已经不满足键盘的操作方式,而是更加倾心于使用手指在屏幕上进行操作。

Android 界面框架支持对触摸事件的监听,并能够将触摸事件的详细信息传递给处理函数。为了处理控件的触摸事件,先需要设置触摸事件的监听器,并重载 onTouch()

函数。示例代码如下：

```
1   touchView.setOnTouchListener(new View.OnTouchListener(){
2       @Override
3       public boolean onTouch(View v, MotionEvent event) {
4           //过程代码…
5           return true/false;
6       }
```

第1行代码是设置控件的触摸事件监听器。在代码第3行的onTouch()函数中，第1个参数View表示产生触摸事件的界面控件；第2个参数MontionEvent表示触摸事件的详细信息，如产生时间、坐标和触点压力等。第5行是onTouch()函数的返回值。

TouchEventDemo是一个说明如何处理触摸事件的示例。在这个示例中，用户界面如图5.32所示。

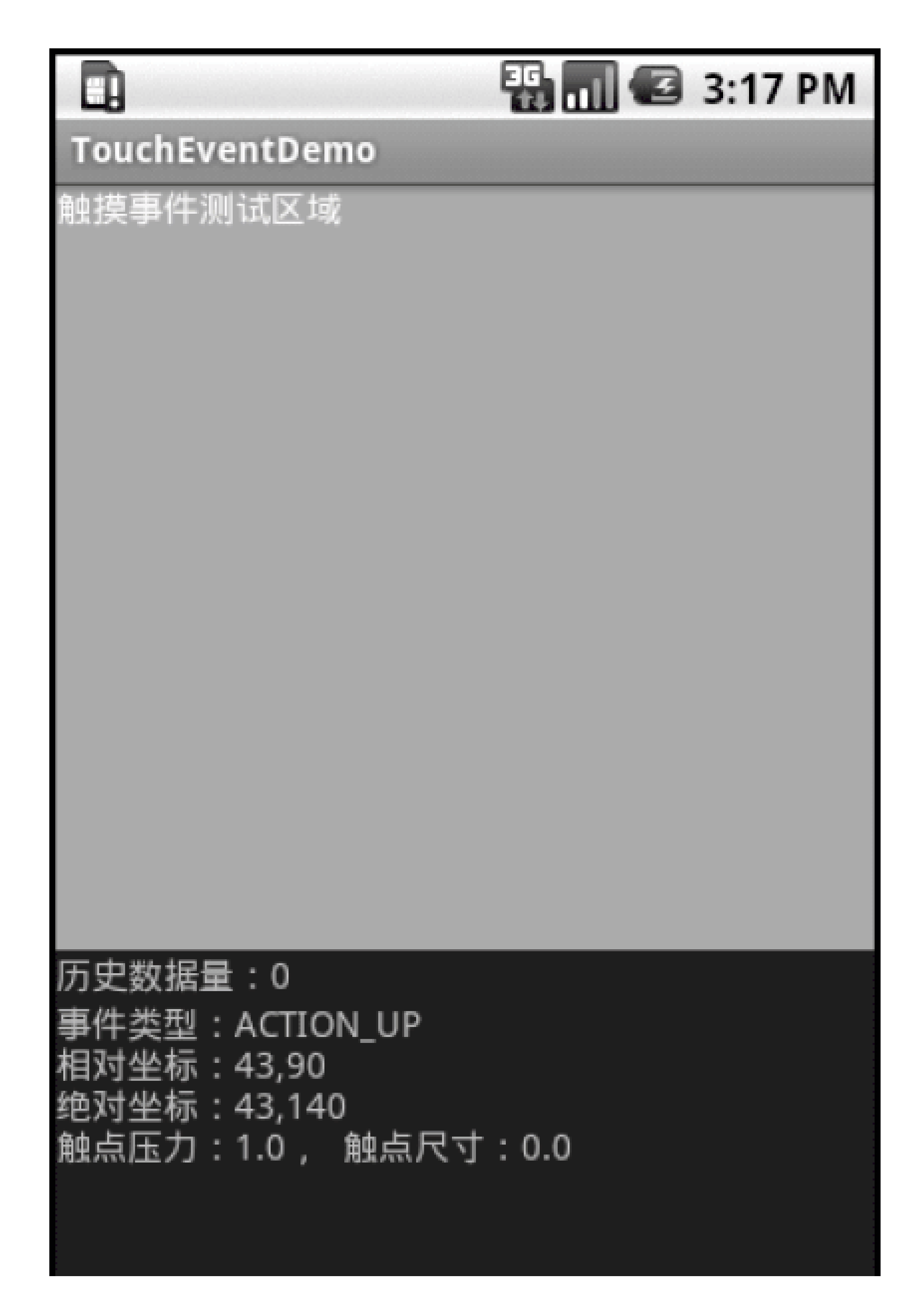



图5.32 TouchEventDemo界面

在TouchEventDemo的用户界面中，浅蓝色区域是可以接受触摸事件的区域，用户可以在Android模拟器中使用鼠标点击屏幕，用以模拟触摸手机屏幕。下方黑色区域是显示区域，用来显示触摸事件的类型、相对坐标、绝对坐标、触点压力、触点尺寸和历史数据量等信息。

在用户界面中使用了线性布局，并加入了3个TextView控件，第1个TextView(ID为touch_area)用来标识触摸事件的测试区域，第2个TextView(ID为history_label)用来显示触摸事件的历史数据量，第3个TextView(ID为event_label)用来触摸事件的详细信息，包括类型、相对坐标、绝对坐标、触点压力和触点尺寸。XML文件的代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView  android:id="@+id/touch_area"
        android:layout_width="fill_parent"
        android:layout_height="300dip"
        android:background="#80A0FF"
        android:textColor="#FFFFFF"
        android:text="触摸事件测试区域">
    </TextView>
    <TextView android:id="@+id/history_label"
        android:layout_width="wrap_content"
```

```
15              android:layout_height="wrap_content"
16              android:text="历史数据量：">
17          </TextView>
18          <TextView android:id="@+id/event_label"
19              android:layout_width="wrap_content"
20              android:layout_height="wrap_content"
21              android:text="触摸事件"">
22          </TextView>
23      </LinearLayout>
```

上面的代码中，第 9 行定义了 TextView 的背景颜色，#80A0FF 是颜色代码。第 10 行定义了 TextView 的字体颜色。

在代码中为了能够引用 XML 文件中声明的界面元素，使用了下面的代码：

```
TextView labelView=null;
labelView=(TextView)findViewById(R.id.event_label);
TextView touchView=(TextView)findViewById(R.id.touch_area);
final TextView historyView=(TextView)findViewById(R.id.history_label);
```

当手指接触到触摸屏、在触摸屏上移动或离开触摸屏时，分别会引发 ACTION_DOWN、ACTION_UP 和 ACTION_MOVE 触摸事件，而无论是哪种触摸事件，都会调用 onTouch()函数进行处理。事件类型包含在 onTouch()函数的 MotionEvent 参数中，可以通过 getAction()函数获取到触摸事件的类型，然后根据触摸事件的不同类型进行不同的处理。但为了能够使屏幕最上方的 TextView 处理触摸事件，需要使用 setOnTouchListener()函数在代码中设置触摸事件监听器，并在 onTouch()函数添加触摸事件的处理过程。

```
touchView.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                int action=event.getAction();
                switch (action) {
                  case (MotionEvent.ACTION_DOWN):
                      Display("ACTION_DOWN",event);
                      break;
                  case (MotionEvent.ACTION_UP):
                      int historySize=ProcessHistory(event);
                      historyView.setText("历史数据量："+historySize);
                      Display("ACTION_UP",event);
                      break;
                  case (MotionEvent.ACTION_MOVE):
                      Display("ACTION_MOVE",event);
                      break;
                }
                return true;
```

```
19                      }
20              });
```

第 7 行代码的 Display()是一个自定义函数,主要用来显示触摸事件的详细信息,函数的代码和含义将在后面进行介绍。第 10 行代码的 ProcessHistory()也是一个自定义函数,用来处理触摸事件的历史数据,后面进行介绍。第 11 行代码是使用 TextView 显示历史数据的数量。

MotionEvent 参数中不仅有触摸事件的类型信息,还有触点的坐标信息,获取方法是使用 getX()和 getY()函数,这两个函数获取到的是触点相对于父界面元素的坐标信息。如果需要获取绝对坐标信息,则可使用 getRawX()和 getRawY()函数。触点压力是一个介于 0 和 1 之间的浮点数,用来表示用户对触摸屏施加压力的大小,接近 0 表示压力较小,接近 1 表示压力较大,获取触摸事件触点压力的方式是调用 getPressure()函数。触点尺寸指用户接触触摸屏的接触点大小,也是一个介于 0 和 1 之间的浮点数,接近 0 表示尺寸较小,接近 1 表示尺寸较大,可以使用 getSize()函数获取。

Display()将 MotionEvent 参数中的事件信息提取出来,并显示在用户界面上。

```
private void Display(String eventType, MotionEvent event){
        int x= (int)event.getX();
        int y= (int)event.getY();
        float pressure= event.getPressure();
        float size= event.getSize();
        int RawX= (int)event.getRawX();
        int RawY= (int)event.getRawY();

        String msg= "";
        msg += "事件类型: "+ eventType + "\n";
        msg += "相对坐标: "+ String.valueOf(x)+ ","+ String.valueOf(y)+ "\n";
        msg += "绝对坐标: "+ String.valueOf(RawX)+ ","+ String.valueOf(RawY)+ "\n";
        msg += "触点压力: "+ String.valueOf(pressure)+ ",   ";
        msg += "触点尺寸: "+ String.valueOf(size)+ "\n";
        labelView.setText(msg);
}
```

一般情况下,如果用户将手指放在触摸屏上,但不移动,然后抬起手指,应先后产生 ACTION_DOWN 和 ACTION_UP 两个触摸事件。但如果用户在屏幕上移动手指,然后再抬起手指,则会产生这样的事件序列 ACTION_DOWN→ACTION_MOVE→ACTION_MOVE→ACTION_MOVE→…→ACTION_UP。

在手机上运行的应用程序,效率是非常重要的。如果 Android 界面框架不能产生足够多的触摸事件,则应用程序就不能够很精确地描绘触摸屏上的触摸轨迹。相反,如果 Android 界面框架产生了过多的触摸事件,虽然能够满足精度的要求,但却降低了应用程序效率。Android 界面框架使用了“打包”的解决方法。在触点移动速度较快时会产生大量的数据,每经过一定的时间间隔便会产生一个 ACTION_MOVE 事件,在这个事件中,除了有当前触点的相关信息外,还包含这段时间间隔内触点轨迹的历史数据信息,这样

既能够保持精度，又不至于产生过多的触摸事件。通常情况下，在 ACTION_MOVE 的事件处理函数中，都先处理历史数据，然后再处理当前数据。

```
1   private int ProcessHistory(MotionEvent event)
2       {
3           int historySize=event.getHistorySize();
4           for (int i=0; i<historySize; i++) {
5               long time=event.getHistoricalEventTime(i);
6               float pressure=event.getHistoricalPressure(i);
7               float x=event.getHistoricalX(i);
8               float y=event.getHistoricalY(i);
9               float size=event.getHistoricalSize(i);
10
11              //处理过程…
12          }
13          return historySize;
14      }
```

在 ProcessHistory()函数中，第 3 行代码获取了历史数据的数量，然后在第 4 行至 12 行中循环处理这些历史数据。第 5 行代码获取了历史事件的发生时间，第 6 行代码获取历史事件的触点压力，第 7 行和第 8 行代码获取历史事件的相对坐标，第 9 行获取历史事件的触点尺寸。在第 14 行返回历史数据的数量，主要是用于界面显示。

非常可惜，Android 模拟器并不支持触点压力和触点尺寸的模拟，所有触点压力恒为 1.0，触点尺寸恒为 0.0。同时，作者在 Android 模拟器上也无法产生历史数据，因此历史数据量一直显示为 0。

## 习　题

1. 简述 5 种界面布局的特点。

2. 参考下图中界面控件的摆放位置，分别使用线性布局、相对布局和绝对布局实现用户界面，并对比各种布局实现的复杂程度和对不同屏幕尺寸的适应能力。




3. 简述 Android 系统支持的三种菜单。

4. EditText 控件具有 Numeric 属性，设置成 integer 后 EditText 控件中只能输入数字，而无法输入其他字母或符号。利用按键事件，编程实现 EditText 控件的这一功能。

# 第 6 章 chapter 6

# 组件通信与广播消息

Intent 是轻量级的进程间通信机制，用于跨进程的组件通信和发送系统级广播。通过本章的学习可以让读者基本了解 Android 系统的组件通信原理，掌握利用组件通信启动其他组件的方法，以及利用组件通信获取信息和发送广播消息的方法。

**本章学习目标：**

- 了解使用 Intent 进行组件通信的原理
- 掌握使用 Intent 启动 Activity 的方法
- 掌握获取 Activity 返回值的方法
- 了解 Intent 过滤器的原理与匹配机制
- 掌握发送和接收广播消息的方法

## 6.1 Intent

Intent 是一个动作的完整描述，包含了动作的产生组件、接收组件和传递的数据信息。当然，也可以将 Intent 视为一个在不同组件之间传递的消息，这个消息在到达接收组件后，接收组件会执行相关的动作。

由于 Intent 的存在，使得 Android 系统中互相独立的应用程序组件，成为了一个可以互相通信的组件集合。因此，无论这些组件是否在同一个应用程序中，Intent 可以将一个组件的数据和动作传递给另一个组件。

Intent 为 Activity、Service 和 BroadcastReceiver 等组件提供交互能力。Intent 的一个最常见的用途就是启动 Activity 和 Service，另一个用途是在 Android 系统上发布广播消息。广播消息可以是接收到特定数据或消息，也可以是手机的信号变化或电池的电量过低等信息。

### 6.1.1 启动 Activity

在 Android 系统中，应用程序一般都有多个 Activity，Intent 可以实现不同 Activity 之间的切换和数据传递。Intent 启动 Activity 方式可以分为显式启动和隐式启动。显式启动必须在 Intent 中指明启动的 Activity 所在的类，而隐式启动则由 Android 系统，根

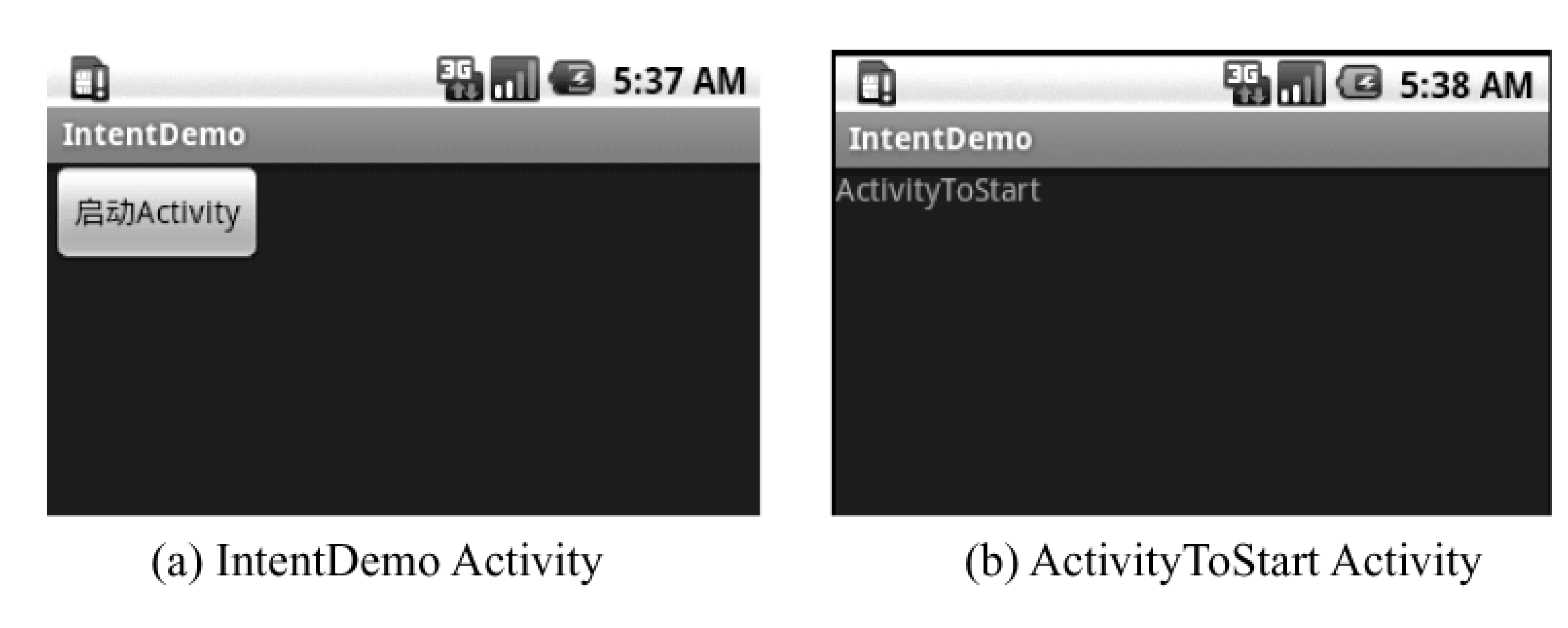
据 Intent 的动作和数据来决定启动哪一个 Activity。也就是说在隐式启动时,Intent 中只包含需要执行的动作和所包含的数据,而无需指明具体启动哪一个 Activity,选择权由 Android 系统和最终用户来决定。

### 1. 显式启动

使用 Intent 来显式启动 Activity,首先需要创建一个 Intent,并为它指定当前的应用程序上下文以及要启动的 Activity,把创建好的这个 Intent 作为参数传递给 startActivity()方法。

```
Intent intent=new Intent(IntentDemo.this, ActivityToStart.class);
startActivity(intent);
```

下面用 IntentDemo 示例说明如何使用 Intent 启动新的 Activity。IntentDemo 示例包含两个 Activity 类,分别是 IntentDemo 和 ActivityToStart。程序启动是默认启动 IntentDemo 这个 Activity,在用户单击“启动 Activity”按钮后,程序启动 ActivityToStart 这个 Activity,如图 6.1 所示。




(a) IntentDemo Activity (b) ActivityToStart Activity

**图 6.1 IntentDemo 示例用户界面**

在 IntentDemo 示例中使用了两个 Activity,因此需要在 AndroidManifest.xml 文件中注册这两个 Activity。注册 Activity 应使用<activity>标签,嵌套在<application>标签内部。

AndroidManifest.xml 文件代码如下:

```
<?xml version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hrbeu.IntentDemo"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"android:label="@string/app_name">
        <activity android:name=".IntentDemo"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
```

```
        </activity>
        <activity android:name=".ActivityToStart"
                  android:label="@string/app_name">
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3"/>
</manifest>
```

Android 应用程序中,用户使用的每个组件都必须在 AndroidManifest.xml 文件中的<application>节点内定义。在上面的代码中,<application>节点下共有两个<activity>节点,分别代表应用程序中所使用的两个 Activity,即 IntentDemo 和 ActivityToStart。

在 IntentDemo.java 文件中,包含了显示使用 Intent 启动 Activity 的核心代码:

```
Button button=(Button)findViewById(R.id.btn);
button.setOnClickListener(new OnClickListener(){
    public void onClick(View view){
        Intent intent=new Intent(IntentDemo.this, ActivityToStart.class);
        startActivity(intent);
    }
});
```

在点击事件的处理函数中,Intent 构造函数的第 1 个参数是应用程序上下文,程序中的应用程序上下文就是 IntentDemo;第 2 个参数是接收 Intent 的目标组件,使用的是显式启动方式,直接指明了需要启动的 Activity。

### 2. 隐式启动

隐式启动的好处在于不需要指明需要启动哪一个 Activity,而由 Android 系统来决定,这样有利于使用第三方组件。

隐式启动 Activity 时,Android 系统在应用程序运行时解析 Intent,并根据一定的规则对 Intent 和 Activity 进行匹配,使 Intent 上的动作、数据与 Activity 完全吻合。匹配的 Activity 可以是应用程序本身的,也可以是 Android 系统内置的,还可以是第三方应用程序提供的。因此,这种方式更加强调了 Android 应用程序中组件的可复用性。

例如,用户如果希望启动一个浏览器,查看指定的网页内容,却不能确定具体应该启动哪一个 Activity,此时则可以使用 Intent 的隐式启动方式,由 Android 系统在程序运行时决定具体启动哪一个应用程序的 Activity 来接收这个 Intent。程序开发人员可以将浏览动作和 Web 地址作为参数传递给 Intent,Android 系统则通过匹配动作和数据格式,找到最适合于此动作的组件。在默认情况下,Android 系统会调用内置的 Web 浏览器。

```
Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
startActivity(intent);
```

Intent 的动作是 Intent. ACTION_VIEW，根据 URI 的数据类型来匹配动作。数据部分的 URI 是 Web 地址，使用 Uri. parse(urlString)方法，可以简单地把一个字符串解释成 Uri 对象。

Intent 的语法如下：

```
Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(urlString));
```

Intent 构造函数的第 1 个参数是 Intent 需要执行的动作，Android 系统支持的常见动作字符串常量可以参考表 6.1。第 2 个参数是 URI，表示需要传递的数据。

**表 6.1　Intent 常用动作**

| 动　　作 | 说　　明 |
| --- | --- |
| ACTION_ANSWER | 打开接听电话的 Activity，默认为 Android 内置的拨号盘界面 |
| ACTION_CALL | 打开拨号盘界面并拨打电话，使用 Uri 中的数字部分作为电话号码 |
| ACTION_DELETE | 打开一个 Activity，对所提供的数据进行删除操作 |
| ACTION_DIAL | 打开内置拨号盘界面，显示 Uri 中提供的电话号码 |
| ACTION_EDIT | 打开一个 Activity，对所提供的数据进行编辑操作 |
| ACTION_INSERT | 打开一个 Activity，在提供数据的当前位置插入新项 |
| ACTION_PICK | 启动一个子 Activity，从提供的数据列表中选取一项 |
| ACTION_SEARCH | 启动一个 Activity，执行搜索动作 |
| ACTION_SENDTO | 启动一个 Activity，向数据提供的联系人发送信息 |
| ACTION_SEND | 启动一个可以发送数据的 Activity |
| ACTION_VIEW | 最常用的动作，对以 Uri 方式传送的数据，根据 Uri 协议部分以最佳方式启动相应的 Activity 进行处理。对于 http:address 将打开浏览器查看；对于 tel:address 将打开拨号呼叫指定的电话号码 |
| ACTION_WEB_SEARCH | 打开一个 Activity，对提供的数据进行 Web 搜索 |

WebViewIntentDemo 示例说明了如何隐式启动 Activity，用户界面如图 6.2(a)所示。

当用户在文本框中输入要访问网址后，通过单击“浏览此 URL”按钮，程序根据用户输入的网址生成一个 Intent，并以隐式启动的方式调用 Android 内置的 Web 浏览器，并打开指定的 Web 页面。本例输入的网址是哈尔滨工程大学的主站地址，地址是 http://www. hrbeu. edu. cn，打开页面后的效果如图 6.2(b)所示。

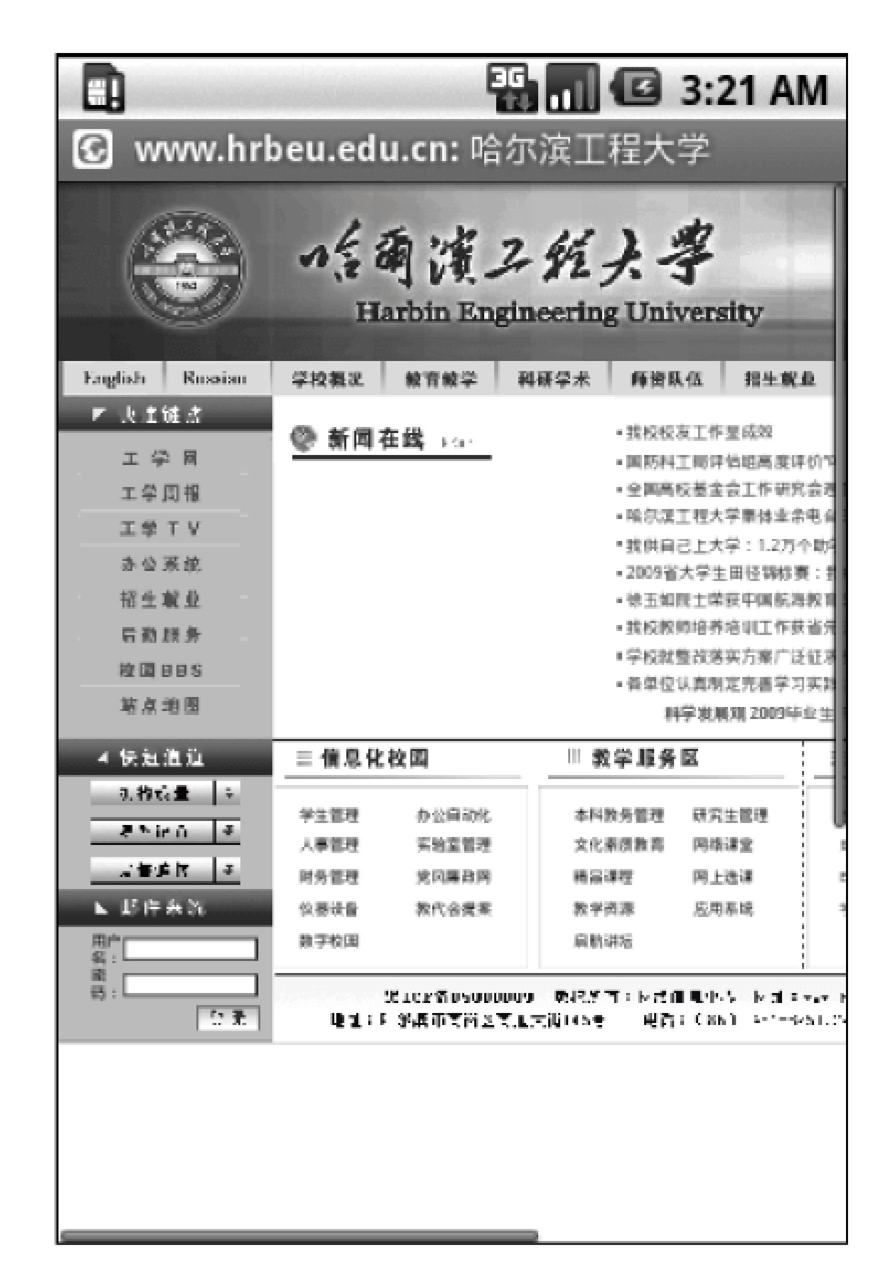
### 6.1.2　获取 Activity 返回值

在 6.1.1 节 IntentDemo 示例中，通过使用 startActivity(Intent)方法启动 Activity，启动后的两个 Activity 之间相互独立，没有任何的关联。在很多情况下，后启动的 Activity 是为了让用户对特定信息进行选择，在关闭这个 Activity 后，用户的选择信息需

要返回给未关闭的那个 Activity。按照 Activity 启动的先后顺序，先启动的称为父 Activity，后启动的称为子 Activity。如果需要将子 Activity 的部分信息返回给父 Activity，则可以使用 Sub-Activity 的方式去启动子 Activity。




(a) 输入网址的界面




(b) 打开 Web 页面后的界面

**图 6.2 WebViewIntentDemo 用户界面**

获取子 Activity 的返回值，一般可以分为以下三个步骤：

(1) 以 Sub-Activity 的方式启动子 Activity。

(2) 设置子 Activity 的返回值。

(3) 在父 Activity 中获取返回值。

下面详细介绍每一个步骤的过程和代码实现。

### 1. 以 Sub-Activity 的方式启动子 Activity

以 Sub-Activity 方式启动子 Activity，需要调用 startActivityForResult(Intent，requestCode)函数，参数 Intent 用于决定启动哪个 Activity，参数 requestCode 是唯一的标识子 Activity 的请求码。因为所有子 Activity 返回时，父 Activity 都调用相同的处理函数，因此父 Activity 使用 requestCode 来确定是哪一个子 Activity 的返回值。

显式启动子 Activity 的代码如下：

```
int SUBACTIVITY1=1;
Intent intent=new Intent(this, SubActivity1.class);
startActivityForResult(intent, SUBACTIVITY1);
```

隐式启动子 Activity 的代码如下：

```
int SUBACTIVITY2=2;
Uri uri=Uri.parse("content://contacts/people");
```

```
Intent intent=new Intent(Intent.ACTION_PICK, uri);
startActivityForResult(intent, SUBACTIVITY2);
```

### 2. 设置子 Activity 的返回值

在子 Activity 调用 finish()函数关闭前，调用 setResult()函数将所需的数据返回给父 Activity。setResult()函数有两个参数，一个是结果码，一个是返回值。结果码表明了子 Activity 的返回状态，通常为 Activity.RESULT_OK 或者 Activity.RESULT_CANCELED，也可以是自定义的结果码，结果码均为整数类型。返回值封装在 Intent 中，也就是说子 Activity 通过 Intent 将需要返回的数据传递给父 Activity。数据主要以 Uri 形式提供的返回给父 Activity，此外还可以附加一些额外信息，这些额外信息用 Extra 的集合表示。

以下代码说明如何在子 Activity 中设置返回值：

```
Uri data=Uri.parse("tel:"+tel_number);
Intent result=new Intent(null, data);
result.putExtra("address", "JD Street");
setResult(RESULT_OK, result);
finish();
```

### 3. 在父 Activity 中获取返回值

当子 Activity 关闭时，启动它的父 Activity 的 onActivityResult()函数将被调用。如果需要在父 Activity 中处理子 Activity 的返回值，则重载此函数即可。此函数的语法如下：

```
public void onActivityResult(int requestCode, int resultCode, Intent data);
```

其中第 1 个参数 requestCode，用来表示是哪一个子 Activity 的返回值，第 2 个参数 resultCode 用于表示子 Activity 的返回状态，第 3 个参数 data 是子 Activity 的返回数据，返回数据类型是 Intent。根据返回数据的用途不同，Uri 数据的协议则不同，也可以使用 Extra 方法返回一些原始类型的数据。

以下代码说明如何在父 Activity 中处理子 Activity 的返回值：

```
private static final int SUBACTIVITY1=1;
private static final int SUBACTIVITY2=2;

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data){
    Super.onActivityResult(requestCode, resultCode, data);
    switch(requestCode){
        case SUBACTIVITY1:
            if (resultCode==Activity.RESULT_OK){
```

```
10                    Uri uriData=data.getData();
11                }else if (resultCode ==Activity.RESULT_CANCEL){
12                }
13            break;
14            case SUBACTIVITY2:
15                if (resultCode ==Activity.RESULT_OK){
16                    Uri uriData=data.getData();
17                }
18            break;
19        }
20    }
```

代码的第 1 行和第 12 行是两个子 Activity 的请求码，在第 7 行对请求码进行匹配。代码第 9 行和第 11 行对结果码进行判断，如果返回的结果码是 Activity.RESULT_OK，则在代码的第 10 行使用 getData()函数获取 Intent 中的 Uri 数据；如果返回的结果码是 Activity.RESULT_CANCELED，则不进行任何操作。




**图 6.3　ActivityCommunication 用户界面**

ActivityCommunication 示例说明了如何以 Sub-Activity 方式启动子 Activity，以及如何使用 Intent 进行组件间通信，该示例的用户界面如图 6.3 所示。

当用户单击“启动 Activity1”和“启动 Activity2”按钮时，程序将分别启动子 SubActivity1 和 SubActivity2，如图 6.4 所示。SubActivity1 提供了一个输入框，以及“接受”和“撤销”两个按钮。如果在输入框中输入信息后单击“接受”按钮，程序会把输入框中的信息传递给其父 Activity，并在父 Activity 的界面上显示。而如果用户单击“撤销”按钮，则程序不会向父 Activity 传递任何信息。SubActivity2 主要是为了说明如何在父 Activity 中处理多个子 Activity，因此仅提供了用于关闭 SubActivity2 的“关闭”按钮。




(a) SubActivity1　　(b) SubActivity2

**图 6.4　ActivityCommunication 的两个子 Activity**

ActivityCommunication 示例的文件结构如图 6.5 所示，父 Activity 的代码在 ActivityCommunication.java 文件中，界面布局在 main.xml 中；两个子 Activity 的代码

分别在 SubActivity1. java 和 SubActivity2. java 文件中，界面布局分别在 subactivity1. xml 和 subactivity2. xml 中。

ActivityCommunication. java 文件的核心代码如下：

```
ActivityCommunication
  Android 1.5
  src
    edu.hrbeu.ActivityCommunication
      ActivityCommunication.java
      SubActivity1.java
      SubActivity2.java
  gen [Generated Java Files]
  assets
  res
    drawable
    layout
      main.xml
      subactivity1.xml
      subactivity2.xml
    values
  AndroidManifest.xml
  default.properties
```

图 6.5 ActivityCommunication 文件结构

```java
public class ActivityCommunication extends Activity {
    private static final int SUBACTIVITY1=1;
    private static final int SUBACTIVITY2=2;
    TextView textView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView=(TextView)findViewById(R.id.textShow);
        final Button btn1=(Button)findViewById(R.id.btn1);
        final Button btn2=(Button)findViewById(R.id.btn2);

        btn1.setOnClickListener(new OnClickListener(){
            public void onClick(View view){
                Intent intent=new Intent(ActivityCommunication.this, SubActivity1.class);
                startActivityForResult(intent, SUBACTIVITY1);
            }
        });

        btn2.setOnClickListener(new OnClickListener(){
            public void onClick(View view){
            Intent intent=new Intent(ActivityCommunication.this, SubActivity2.class);
                startActivityForResult(intent, SUBACTIVITY2);
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        switch(requestCode){
        case SUBACTIVITY1:
            if (resultCode==RESULT_OK){
                Uri uriData=data.getData();
                textView.setText(uriData.toString());
```

```
37                    }
38                    break;
39                case SUBACTIVITY2:
40                    break;
41                }
42            }
43    }
```

在代码的第 2 行和第 3 行分别定义了两个子 Activity 的请求码。在代码的第 16 行和第 23 行以 Sub-Activity 的方式分别启动两个子 Activity。代码第 29 行是子 Activity 关闭后的返回值处理函数，其中 requestCode 是子 Activity 返回的请求码，应该与第 2 行和第 3 行定义的两个请求码相匹配；resultCode 是结果码，在代码第 32 行对结果码进行判断，如果等于 RESULT_OK，在第 35 行代码获取子 Activity 的返回值中的数据；data 是返回值，子 Activity 需要返回的数据就保存在 data 中。

SubActivity1.java 的核心代码如下：

```
public class SubActivity1 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.subactivity1);
        final EditText editText= (EditText)findViewById(R.id.edit);
        Button btnOK= (Button)findViewById(R.id.btn_ok);
        Button btnCancel= (Button)findViewById(R.id.btn_cancel);

        btnOK.setOnClickListener(new OnClickListener(){
            public void onClick(View view){
                String uriString= editText.getText().toString();
                Uri data= Uri.parse(uriString);
                Intent result= new Intent(null, data);
                setResult(RESULT_OK, result);
                finish();
            }
        });

        btnCancel.setOnClickListener(new OnClickListener(){
            public void onClick(View view){
                setResult(RESULT_CANCELED, null);
                finish();
            }
        });
    }
}
```

代码第 13 行将 EditText 控件的内容作为数据保存在 Uri 中，并在第 14 行代码中使用这个 Uri 构造 Intent，在第 15 行代码中，将 Intent 作为返回值，RESUIT_OK 作为结果码，通过调用 setResult()函数，将返回值和结果码传递给父 Activity。最后在代码第 16 行调用 finish()函数关闭当前的子 Activity。

SubActivity2. java 的核心代码如下：

```
1    public class SubActivity2 extends Activity {
2        @Override
3        public void onCreate(Bundle savedInstanceState) {
4            super.onCreate(savedInstanceState);
5            setContentView(R.layout.subactivity2);
6
7            Button btnReturn= (Button)findViewById(R.id.btn_return);
8            btnReturn.setOnClickListener(new OnClickListener(){
9                public void onClick(View view){
10                   setResult(RESULT_CANCELED, null);
11                   finish();
12               }
13           });
14       }
15   }
```

在 SubActivity2 的代码中，第 10 行的 setResult()函数仅设置了结果码，第 2 个参数为 null，表示数据需要传递给父 Activity。

## 6.2 Intent 过滤器

在隐式启动 Activity 时，并没有在 Intent 中指明 Activity 所在的类，因此，Android 一定存在某种匹配机制，使 Android 系统能够根据在 Intent 中的数据信息，找到需要启动的 Activity。这种匹配机制的实现就靠 Android 系统中的 Intent 过滤器（Intent Filter）实现的。

Intent 过滤器是一种根据 Intent 中的动作（Action）、类别（Categorie）和数据（Data）等内容，对适合接收该 Intent 的组件进行匹配和筛选的机制。Intent 过滤器还可以匹配数据类型、路径和协议，还包括可以用来确定多个匹配项顺序的优先级（Priority）。应用程序的 Activity 组件、Service 组件和 BroadcastReceiver 都可以注册 Intent 过滤器。这样，这些组件在特定的数据格式上则可以产生相应的动作。

为了使组件能够注册 Intent 过滤器，通常在 AndroidManifest. xml 文件的各个组件的节点下定义<intent-filter>节点，然后在<intent-filter>节点中声明该组件所支持的动作、执行的环境和数据格式等信息。当然，也可以在程序代码中动态地为组件设置 Intent 过滤器。<intent-filter>节点支持<action>标签、<category>标签和<data>标签，分别用来定义 Intent 过滤器的“动作”、“类别”和“数据”。<intent-filter>节点支

持的标签和属性说明参考表 6.2。

表 6.2 <intent-filter>节点属性

| 标 签 | 属 性 | 说 明 |
|---|---|---|
| <action> | android:name | 指定组件所能响应的动作，用字符串表示，通常使用 Java 类名和包的完全限定名构成 |
| <category> | android:category | 指定以何种方式去服务 Intent 请求的动作 |
| <data> | Android:host | 指定一个有效的主机名 |
| | android:mimetype | 指定组件能处理的数据类型 |
| | android:path | 有效的 URI 路径名 |
| | android:port | 主机的有效端口号 |
| | android:scheme | 所需要的特定的协议 |

<category>标签用来指定 Intent 过滤器的服务方式，每个 Intent 过滤器可以定义多个<category>标签，程序开发人员可以使用自定义的类别，或使用 Android 系统提供的类别。Android 系统提供的类别可以参考表 6.3。

表 6.3 Android 系统提供的类别

| 值 | 说 明 |
|---|---|
| ALTERNATIVE | Intent 数据默认动作的一个可替换的执行方法 |
| SELECTED_ALTERNATIVE | 和 ALTERNATIVE 类似，但替换的执行方法不是指定的，而是被解析出来的 |
| BROWSABLE | 声明 Activity 可以由浏览器启动 |
| DEFAULT | 为 Intent 过滤器中定义的数据提供默认动作 |
| HOME | 设备启动后显示的第一个 Activity |
| LAUNCHER | 在应用程序启动时首先被显示 |

AndroidManifest.xml 文件中的每个组件的<intent-filter>都被解析成一个 Intent 过滤器对象。当应用程序安装到 Android 系统时，所有的组件和 Intent 过滤器都会注册到 Android 系统中。这样，Android 系统便知道了如何将任意一个 Intent 请求通过 Intent 过滤器映射到相应的组件上。

这种 Intent 到 Intent 过滤器的映射过程称为“Intent 解析”。Intent 解析可以在所有的组件中，找到一个可以与请求的 Intent 达成最佳匹配的 Intent 过滤器。Android 系统中 Intent 解析的匹配规则如下：

(1) Android 系统把所有应用程序包中的 Intent 过滤器集合在一起，形成一个完整的 Intent 过滤器列表。

(2) 在 Intent 与 Intent 过滤器进行匹配时，Android 系统会将列表中所有 Intent 过滤器的“动作”和“类别”与 Intent 进行匹配，任何不匹配的 Intent 过滤器都将被过滤掉。

没有指定"动作"的 Intent 过滤器可以匹配任何的 Intent，但是没有指定"类别"的 Intent 过滤器只能匹配没有"类别"的 Intent。

（3）把 Intent 数据 Uri 的每个子部与 Intent 过滤器的<data>标签中的属性进行匹配，如果<data>标签指定了协议、主机名、路径名或 MIME 类型，那么这些属性都要与 Intent 的 Uri 数据部分进行匹配，任何不匹配的 Intent 过滤器均被过滤掉。

（4）如果 Intent 过滤器的匹配结果多于一个，则可以根据在<intent-filter>标签中定义的优先级标签来对 Intent 过滤器进行排序，优先级最高的 Intent 过滤器将被选择。

IntentResolutionDemo 示例说明了如何在 AndroidManifest.xml 文件中注册 Intent 过滤器，以及如何设置<intent-filter>节点属性来捕获指定的 Intent。AndroidManifest.xml 的完整代码如下：

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        package="edu.hrbeu.IntentResolutionDemo"
4        android:versionCode="1"
5        android:versionName="1.0">
6        <application android:icon="@drawable/icon" android:label="@string/app_name">
7            <activity android:name=".IntentResolutionDemo"
8                    android:label="@string/app_name">
9                <intent-filter>
10                   <action android:name="android.intent.action.MAIN"/>
11                   <category android:name="android.intent.category.LAUNCHER"/>
12               </intent-filter>
13           </activity>
14           <activity android:name=".ActivityToStart"
15                   android:label="@string/app_name">
16               <intent-filter>
17                   <action android:name="android.intent.action.VIEW"/>
18                   <category android:name="android.intent.category.DEFAULT"/>
19                   <data android:scheme="schemodemo" android:host="edu.hrbeu"/>
20               </intent-filter>
21           </activity>
22       </application>
23       <uses-sdk android:minSdkVersion="3"/>
24   </manifest>
```

在代码的第 7 行和第 14 行分别定义了两个 Activity。第 9 行到第 12 行是第 1 个 Activity 的 Intent 过滤器，动作是 android.intent.action.MAIN，类别是 android.intent.category.LAUNCHER。由过滤器的动作和类别可知，这个 Activity 是应用程序启动后显示的默认用户界面。

第 16～20 行是第 2 个 Activity 的 Intent 过滤器，过滤器的动作是 android.intent.action.VIEW，表示根据 Uri 协议，以最佳的方式启动相应的 Activity；类别是 android.

intent. category. DEFAULT,表示数据的默认动作;数据的协议部分是 android:scheme="schemodemo",数据的主机名称部分是 android:host="edu. hrbeu"。

在 IntentResolutionDemo. java 文件中,定义了一个 Intent 用来启动一个新的 Activity,这个 Intent 与 Activity 设置的 Intent 过滤器是完全匹配的。IntentResolutionDemo. java 文件中 Intent 实例化和启动 Activity 的代码如下:

```
1    Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse("schemodemo://edu.hrbeu/path"));
2    startActivity(intent);
```

代码第 1 行所定义的 Intent,动作为 Intent. ACTION_VIEW,与 Intent 过滤器的动作 android. intent. action. VIEW 匹配;Uri 是"schemodemo://edu. hrbeu/path",其中的协议部分为"schemodemo",主机名部分为"edu. hrbeu",也与 Intent 过滤器定义的数据要求完全匹配。因此,代码第 1 行定义的 Intent,在 Android 系统与 Intent 过滤器列表进行匹配时,会与 AndroidManifest. xml 文件中 ActivityToStart 定义的 Intent 过滤器完全匹配。

## 6.3 广播消息

Intent 的另一种用途是发送广播消息。应用程序和 Android 系统都可以使用 Intent 发送广播消息,广播消息的内容可以是与应用程序密切相关的数据信息,也可以是 Android 的系统信息,例如网络连接变化、电池电量变化、接收到短信和系统设置变化等等。如果应用程序注册了 BroadcastReceiver,则可以接收到指定的广播消息。

使用 Intent 广播消息非常简单,只需创建一个 Intent,并调用 sendBroadcast()函数就可把 Intent 携带的消息广播出去。但需要注意的是,在构造 Intent 时必须用一个全局唯一的字符串标识其要执行的动作,通常使用应用程序包的名称。如果要在 Intent 传递额外数据,可以用 Intent 的 putExtra()方法。下面的代码构造用于广播消息的 Intent,并添加了额外的数据,然后调用 sendBroadcast()发生了广播消息:

```
String UNIQUE_STRING="edu.hrbeu.BroadcastReceiverDemo";
Intent intent=new Intent(UNIQUE_STRING);
intent.putExtra("key1", "value1");
intent.putExtra("key2", "value2");
sendBroadcast(intent);
```

BroadcastReceiver 用于监听广播消息,可以在 AndroidManifest. xml 文件或在代码中注册一个 BroadcastReceiver,并在其中使用 Intent 过滤器指定要处理的广播消息。创建 BroadcastReceiver 需继承 BroadcastReceiver 类,并重载 onReceive()方法。示例代码如下:

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
```

```
            //TODO: React to the Intent received.
        }
    }
```

注册 BroadcastReceiver 的应用程序不需要一直运行，当 Android 系统接收到与之匹配的广播消息时，会自动启动此 BroadcastReceiver，因此 BroadcastReceiver 非常适合做一些资源管理的工作。在 BroadcastReceiver 接收到与之匹配的广播消息后，onReceive()方法会被调用，但 onReceive()方法必须要在 5 秒钟执行完毕，否则 Android 系统会认为该组件失去响应，并提示用户强行关闭该组件。




**图 6.6 BroadcastReceiverDemo 主界面**

BroadcastReceiverDemo 示例说明了如何在应用程序中注册 BroadcastReceiver，并接收指定类型的广播消息。BroadcastReceiverDemo 示例的界面如图 6.6 所示，在点击"发送广播消息"按钮后，EditText 控件中内容将以广播消息的形式发送出去，示例内部的 BroadcastReceiver 将接收这个广播消息，并显示在用户界面的下方。

BroadcastReceiverDemo.java 文件中包含发送广播消息的代码，其关键代码如下：

```
1   button.setOnClickListener(new OnClickListener(){
2       public void onClick(View view){
3           Intent intent=new Intent("edu.hrbeu.BroadcastReceiverDemo");
4           intent.putExtra("message", entryText.getText().toString());
5           sendBroadcast(intent);
6       }
7   });
```

代码第 3 行创建 Intent 时，将 edu.hrbeu.BroadcastReceiverDem 作为识别广播消息的字符串标识，并在代码第 4 行将添加了额外信息，最后在代码第 5 行调用 sendBroadcast()函数发送广播消息。

为了能够使应用程序中的 BroadcastReceiver 接收指定的广播消息，首先要在 AndroidManifest.xml 文件中添加 Intent 过滤器，声明 BroadcastReceiver 可以接收的广播消息。AndroidManifest.xml 文件的完整代码如下：

```
<?xml version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hrbeu.BroadcastReceiverDemo"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"android:label="@string/app_name">
```

```
7           <activity android:name=".BroadcastReceiverDemo"
8                     android:label="@string/app_name">
9               <intent-filter>
10                  <action android:name="android.intent.action.MAIN"/>
11                  <category android:name="android.intent.category.LAUNCHER"/>
12              </intent-filter>
13          </activity>
14          <receiver android:name=".MyBroadcastReceiver">
15              <intent-filter>
16                  <action android:name="edu.hrbeu.BroadcastReceiverDemo"/>
17              </intent-filter>
18          </receiver>
19      </application>
20      <uses-sdk android:minSdkVersion="3"/>
21  </manifest>
```

在代码的第 14 行中创建了一个<receiver>节点，在第 15 行中声明了 Intent 过滤器的动作为“edu.hrbeu.BroadcastReceiverDemo”，这与 BroadcastReceiverDemo.java 文件中 Intent 的动作相一致，表明这个 BroadcastReceiver 可以接收动作为“edu.hrbeu.BroadcastReceiverDemo”的广播消息。

MyBroadcastReceiver.java 文件创建了一个自定义的 BroadcastReceiver，其核心代码如下：

```
1   public class MyBroadcastReceiver extends BroadcastReceiver {
2       @Override
3       public void onReceive(Context context, Intent intent) {
4           String msg=intent.getStringExtra("message");
5           Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
6       }
7   }
```

代码第 1 行首先继承了 BroadcastReceiver 类，并在第 3 行重载了 onReveive()函数。当接收到 AndroidManifest.xml 文件定义的广播消息后，程序将自动调用 onReveive()函数。代码第 4 行通过调用 getStringExtra()函数，从 Intent 中获取标识为 message 的字符串数据，并使用 Toast 将信息显示在屏幕上。第 5 行代码的 Toast 是一个显示提示信息的类，调用 makeText()函数可将提示信息短时间的浮现在用户界面之上。makeText()函数的第 1 个参数是上下文信息；第 2 个参数是需要显示的提示信息；第 3 个参数是显示的时间，Toast.LENGTH_SHORT 表示短时间显示，Toast.LENGTH_LONG 表示长时间显示；最后调用 show()方法将提示信息实际显示在界面之上。

## 习　　题

1. 简述 Intent 的定义和用途。

2. 简述 Intent 过滤器的定义和功能。

3. 简述 Intent 解析的匹配规则。

4. 编程实现下述功能：主界面上有一个“登录”按钮，单击“登录”按钮后打开一个新的 Activity；新的 Activity 上面有输入用户名和密码的控件，在用户关闭这个 Activity 后，将用户输入的用户名和密码传递到主界面中。

# 第 7 章 chapter 7

# 后台服务

Service 是 Android 系统的后台服务组件，适用于开发无界面、长时间运行的应用功能。通过本章的学习可以让读者了解后台服务的基本原理，掌握进程内服务与跨进程服务的使用方法，有助于深入理解 Android 系统的进程间通信机制。

**本章学习目标：**

- 了解 Service 的原理和用途
- 掌握进程内服务的管理方法
- 掌握服务的隐式启动和显式启动方法
- 了解线程的启动、挂起和停止方法
- 了解跨线程的界面更新方法
- 掌握跨进程服务的绑定和调用方法
- 了解 AIDL 语言的用途和语法

## 7.1 Service 简 介

因为手机屏幕尺寸的限制，通常情况下在同一时刻仅有一个应用程序处于激活状态，并能够显示在手机屏幕上。因此，应用程序需要一种机制，在没有用户界面的情况下，使其能够长时间在后台运行，实现应用程序的特定功能，并能够处理事件或更新数据。Android 系统提供的 Service（服务）组件，不直接与用户进行交互，并能够长期在后台运行。有很多情况需要使用 Service，经常提到的例子就是 MP3 播放器，用户需要在关闭播放器界面后，仍能够保持音乐继续播放，这就需要将音乐回放的功能在 Service 组件实现。

Service 非常适用于无需用户干预，且需要长期运行的后台功能。Service 没有用户界面，有利于降低系统资源的消耗，而且 Service 比 Activity 具有更高的优先级，因此在系统资源紧张时，Service 不会轻易被 Android 系统终止。即使 Service 被系统终止，在系统资源恢复后 Service 也将自动恢复运行状态，因此可以认为 Service 是在系统中永久运行的组件。Service 除了可以实现后台服务功能，还可以用于进程间通信（Inter Process Communication，IPC），解决两个不同 Android 应用程序进程之间的调用和通信问题。

Service 的生命周期比较简单，仅包括全生命周期和活动生命周期，还有三个事件回

调函数，分别是 onCreate()、onStart()和 onDestroy()，如图 7.1 所示。

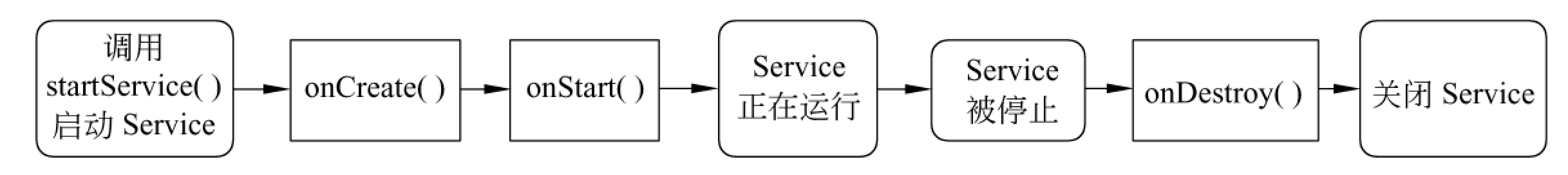



**图 7.1 Service 生命周期**

Service 的生命周期从 onCreate()开始到 onDestroy()结束，在 onCreate()中完成 Service 的初始化工作，在 onDestroy()中释放所有占用的资源。活动生命周期从 onStart()开始，但没有与之对应的“停止”函数，因此可以近似认为活动生命周期也是以 onDestroy()标志结束。

Service 的使用方式一般有两种，一种是启动方式，另一种是绑定方式。在启动方式中，通过调用 Context. startService()启动 Service，通过调用 Context. stopService()或 Service. stopSefl()停止 Service。因此，Service 一定是由其他的组件启动的，但停止过程可以通过其他组件或自身完成。在启动方式中，启动 Service 的组件不能获取到 Service 对象，因此无法调用 Service 中实现的方法，也不能获取 Service 中的任何状态和数据信息。因此，如果仅以启动方式使用 Service，这个 Service 需要具备自管理的能力，且不需要通过函数调用向外部组件提供数据或功能。

在绑定方式中，Service 的使用是通过服务链接(Connection)实现的，服务链接能获取 Service 的对象，因此绑定 Service 的组件可以调用 Service 中实现的函数，或直接获取 Service 中状态和数据信息。使用 Service 的组件通过 Context. bindService()建立服务链接，通过 Context. unbindService()停止服务链接。如果在绑定过程中 Service 没有启动，Context. bindService()会自动启动 Service。而且同一个 Service 可以绑定多个服务链接，这样可以同时为多个不同的组件提供服务。

当然，这两种使用方法并不是完全独立的，可以某些情况下混合使用启动方式和绑定方式。还是以 MP3 播放器为例，在后台工作的 Service 通过 Context. startService()启动某个特定音乐播放，但在播放过程中如果用户需要暂停音乐播放，则需要通过 Context. bindService()获取服务链接和 Service 对象，进而通过调用 Service 的对象中的函数，暂停音乐播放过程，并保存相关信息。在这种情况下，如果调用 Context. stopService()不能停止 Service，需要在所有的服务链接关闭后，Service 才能真正的停止。

## 7.2 进程内服务

### 7.2.1 服务管理

服务管理主要指服务的启动和停止，在介绍如何启动和停止服务前，首先说明如何在代码中实现 Service。Service 是一段在后台运行、没有用户界面的代码，其最小代码集如下：

```
import android.app.Service;
```

```
2   import android.content.Intent;
3   import android.os.IBinder;
4
5   public class RandomService extends Service{
6       @Override
7       public IBinder onBind(Intent intent) {
8          return null;
9       }
10  }
```

在上面的代码中，除了在第 1 行到第 3 行引入必要包外，仅在第 5 行声明了 RandomService 继承了 android.app.Service 类，在第 7 行到第 9 行重载了 onBind()函数。onBind()函数是在 Service 被绑定后调用的函数，能够返回 Service 的对象，在后面的内容中会有详细的介绍。

这个 Service 的最小代码集并不能完成任何实际的功能，为了使 Service 具有实际意义，一般需要重载 onCreate()、onStart()和 onDestroy()。Android 系统在创建 Service 时，会自动调用 onCreate()，用户一般在 onCreate()完成必要的初始化工作，例如创建线程、建立数据库链接等。在 Service 没有必要再存在时，系统会自动调用 onDestroy()，用户在 onDestroy()释放所有占用的资源。通过 Context.startService(Intent)启动 Service 时，onStart()则会被系统调用，Intent 会传递给 Service 一些重要的参数。当然，不是所有的 Service 都需要重载这三个函数，完全可以根据实际情况选择需要重载的某个函数。

```
1   public class RandomService extends Service{
2       @Override
3       public void onCreate() {
4           super.onCreate();
5       }
6       @Override
7       public void onStart(Intent intent, int startId) {
8               super.onStart(intent, startId);
9       }
10      @Override
11      public void onDestroy() {
12                super.onDestroy();
13      }
14  }
```

重载 onCreate()、onStart()和 onDestroy()三个函数时，务必要在代码中调用父函数，如代码的第 4 行、第 8 行和第 12 行。

完成 Service 类后，需要在 AndroidManifest.xml 文件中注册这个 Service。这个注册过程非常重要，如果用户不注册，则这个 Service 根本无法启动。AndroidManifest.xml 文件中注册 Service 的代码如下：

```
<service android:name=".RandomService"/>
```

使用<service>标签声明服务，其中的 android:name 表示的是 Service 的类名称，一定要与用户建立的 Service 类名称一致。

在完成 Service 代码和注册后，下一步来说明如何启动和停止 Service。有两种方法启动 Service，显式启动和隐式启动。显式启动需要在 Intent 中指明 Service 所在的类，并调用 startService(Intent)函数启动 Service，示例代码如下：

```
final Intent serviceIntent=new Intent(this, RandomService.class);
startService(serviceIntent);
```

在上面的代码中，在 Intent 中指明了启动的 Service 在 RandomSerevice.class 中。

隐式启动则需要在注册 Service 时，声明 Intent-filter 的 action 属性。

```
<service android:name=".RandomService">
    <intent-filter>
        <action android:name="edu.hrbeu.RandomService" />
    </intent-filter>
</service>
```

在隐式启动 Service 时，需要设置 Intent 的 action 属性，这样则可以在不声明 Service 所在类的情况下启动服务。隐式启动的代码如下：

```
final Intent serviceIntent=new Intent();
serviceIntent.setAction("edu.hrbeu.RandomService");
```

如果 Service 和调用服务的组件在同一个应用程序中，可以使用显式启动或隐式启动，显式启动更加易于使用，且代码简洁。但如果服务和调用服务的组件在不同的应用程序中，则只能使用隐式启动。

无论是显式启动还是隐式启动，停止 Service 的方法都是相同的，将启动 Service 的 Intent 传递给 stopService(Intent)函数即可，示例代码如下：

```
stopService(serviceIntent);
```

在调用 startService(Intent)函数首次启动 Service 后，系统会先后调用 onCreate()和 onStart()，如果再次调用 startService(Intent)函数，系统则仅调用 onStart()，而不再调用 onCreate()。在调用 stopService(Intent)函数停止 Service 时，系统会调用 onDestroy()。无论调用过多少次 startService(Intent)，在调用 stopService (Intent)函数时，系统仅调用 onDestroy()一次。

SimpleRandomServiceDemo 是在应用程序中建立 Service，并以显式启动服务的示例。在工程中创建了 RandomService 服务，该服务启动后会产生一个随机数，并使用 Toast 显示在屏幕上，如图 7.2 所示。




**图 7.2 SimpleRandomServiceDemo 用户界面**

在界面上“启动 Service”按钮调用 startService(Intent)函数，启动 RandomService 服务。“停止 Service”按钮调用 stopService(Intent)函数，停止 RandomService 服务。为了能够清晰地观察 Service 中 onCreate()、onStart()和 onDestroy()三个函数的调用顺序，在每个函数都中使用 Toast，在界面上产生提示信息。

RandomService.java 文件的代码如下：

```
package edu.hrbeu.SimpleRandomServiceDemo;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class RandomService extends Service{

    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this, "(1) 调用 onCreate()",
                Toast.LENGTH_LONG).show();
    }

    @Override
    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
        Toast.makeText(this, "(2) 调用 onStart()",
                Toast.LENGTH_SHORT).show();

        double randomDouble=Math.random();
        String msg="随机数："+String.valueOf(randomDouble);
        Toast.makeText(this,msg, Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "(3) 调用 onDestroy()",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

在 onStart()函数中添加生产随机数的代码,第 23 行生产一个介于 0~1 之间的随机数,并在第 24 行构造供 Toast 显示的消息。

AndroidManifest. xml 文件的代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="edu.hrbeu.SimpleRandomServiceDemo"
        android:versionCode="1"
        android:versionName="1.0">
    <application android:icon="@drawable/icon"
                        android:label="@string/app_name">
        <activity android:name=".SimpleRandomServiceDemo"
                    android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".RandomService"/>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

在调用 AndroidManifest. xml 文件中,在<application>标签下,包含一个<activity>标签和一个<service>标签,在<service>标签中,声明了 RandomService 所在的类。

SimpleRandomServiceDemo. java 文件的代码如下:

```
package edu.hrbeu.SimpleRandomServiceDemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class SimpleRandomServiceDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button startButton= (Button)findViewById(R.id.start);
        Button stopButton= (Button)findViewById(R.id.stop);
```

```
17            final Intent serviceIntent= new Intent(this, RandomService.class);
18            startButton.setOnClickListener(new Button.OnClickListener(){
19                public void onClick(View view){
20                    startService(serviceIntent);
21                }
22            });
23            stopButton.setOnClickListener(new Button.OnClickListener(){
24                public void onClick(View view){
25                    stopService(serviceIntent);
26                }
27            });
28        }
29    }
```

SimpleRandomServiceDemo.java 文件是应用程序中的 Activity 代码,第 20 行和第 25 行分别是启动和停止 Service 的代码。

下面不再给出隐式启动 Service 的示例代码,请参考 ImplicityRandomServiceDemo 示例。

## 7.2.2 使用线程

在 Android 系统中,Activity、Service 和 BroadcastReceiver 都是工作在主线程上,因此任何耗时的处理过程都会降低用户界面的响应速度,甚至导致用户界面失去响应。当用户界面失去响应超过 5 秒钟,Android 系统会允许用户强行关闭应用程序,如图 7.3 所示。因此,较好的解决方法是将耗时的处理过程转移到子线程上,这样可以避免负责界面更新的主线程无法处理界面事件,从而避免用户界面长时间失去响应。耗时的处理过程除了指运算量巨大的复杂运算外,还包括大量的文件操作、网络操作和数据库操作等等。

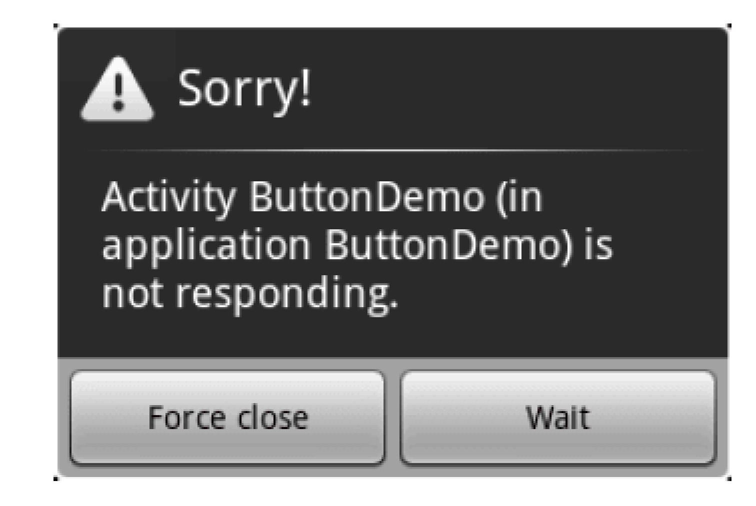



**图 7.3 失去响应时的提示信息**

线程是独立的程序单元,多个线程可以并行工作。在多处理器系统中,每个中央处理器(CPU)单独运行一个线程,因此线程是并行工作的。但在单处理器系统中,处理器会给每个线程一小段时间,在这个时间内线程是被执行的,然后处理器执行下一个线程,这样就产生了线程并行运行的假象。无论线程是否真的并行工作,在宏观上可以认为子线程是独立于主线程,且能与主线程并行工作的程序单元。

在 Java 语言中,建立和使用线程比较简单,首先需要实现 Java 的 Runnable 接口,并重载 run()方法。在 run()中放置代码的主体部分。

```
private Runnable backgroudWork= new Runnable(){
        @Override
        public void run() {
            //过程代码
```

```
    }
};
```

然后创建 Thread 对象,并将上面实现的 Runnable 对象作为参数传递给 Thread 对象。Thread 的构造函数中,第 1 个参数用来表示线程组,第 2 个参数是需要执行的 Runnable 对象,第 3 个参数是线程的名称。

```
private Thread workThread;
workThread= new Thread(null,backgroudWork,"WorkThread");
```

最后,调用 start()方法启动线程。

```
workThread.start();
```

当线程在 run()方法返回后,线程就自动终止了。当然,也可以调用 stop()方法在外部终止线程,但这种方法并不推荐使用,因为这方法并不安全,有可能会产生异常。最好的方法是通知线程自行终止,一般调用 interrupt()方法通告线程准备终止,线程会释放它正在使用的资源,在完成所有的清理工作后自行关闭。

```
workThread.interrupt();
```

其实 interrupt()方法并不能直接终止线程,仅是改变了线程内部的一个布尔字段,run()方法能够检测到这个布尔字段,从而知道何时应该释放资源和终止线程。在 run()方法的代码,一般通过 Thread.interrupted()方法查询线程是否被中断。在很多情况下,子线程需要无限运行,除非外部调用 interrupt()方法中断线程,这样一般会将程序主体放置在 while()内,并调用 Thread.interrupted()方法判断线程是否应被中断。下面的代码中,以 1 秒为间隔循环检测线程是否应被中断。

```
1   public void run() {
2       while(!Thread.interrupted()){
3           //过程代码
4           Thread.sleep(1000);
5       }
6   }
```

第 4 行代码使线程休眠 1000 毫秒。当线程在休眠过程中被中断,则会产生 InterruptedException。在中断的线程上调用 sleep()方法,同样会产生 InterruptedException。因此除了使用 Thread.interrupted()方法判断线程是否应被中断,还可以通过捕获 InterruptedException 判断线程是否应被中断,并且在捕获到 InterruptedException 后,安全终止线程。

```
public void run() {
    try {
        while(true){
            //过程代码
            Thread.sleep(1000);
```

```
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

目前，读者已经可以设计自己的线程，但还存在一个不可回避的问题，即在图形用户界面中使用线程，如何使用线程中的数据更新用户界面。Android 系统提供了多种方法解决这个问题，下面仅介绍如何使用 Handler 更新用户界面。

Handler 允许将 Runnable 对象发送到线程的消息队列中，每个 Handler 对象绑定到一个单独的线程和消息队列上。当用户建立一个新的 Handler 对象，通过 post()方法将 Runnable 对象从后台线程发送到 GUI 线程的消息队列中，当 Runnable 对象通过消息队列后，这个 Runnable 对象将被运行。

```
1    private static Handler handler= new Handler();
2
3    public static void UpdateGUI(double refreshDouble){
4        handler.post(RefreshLable);
5    }
6    private static Runnable RefreshLable= new Runnable(){
7        @Override
8        public void run() {
9            //过程代码
10       }
11   };
```

在上面的代码中，第 1 行建立了一个静态的 Handler 对象，但这个对象是私有的，因此外部代码并不能直接调用这个 Handler 对象。第 3 行 UpdateGUI()是公有的界面更新函数，后台线程通过调用该函数，将后台产生的数据 refreshDouble 传递到 UpdateGUI()函数内部，然后并直接调用 post()方法，将第 6 行的创建的 Runnable 对象传递给界面线程（主线程）的消息队列中。第 7 行到第 10 行代码是 Runnable 对象中需要重载的 run()函数，一般将界面更新代码放置在 run()函数中。

ThreadRandomServiceDemo 是使用线程持续产生随机数的示例。当用户单击“启动 Service”按钮后，将启动后台线程，单击“停止 Service”按钮后，将关闭后台线程。后台线程每 1 秒钟产生一个 0～1 之间的随机数，并通过 Handler 将产生的随机数显示在用户界面上。ThreadRandomServiceDemo 的用户界面如图 7.4 所示。

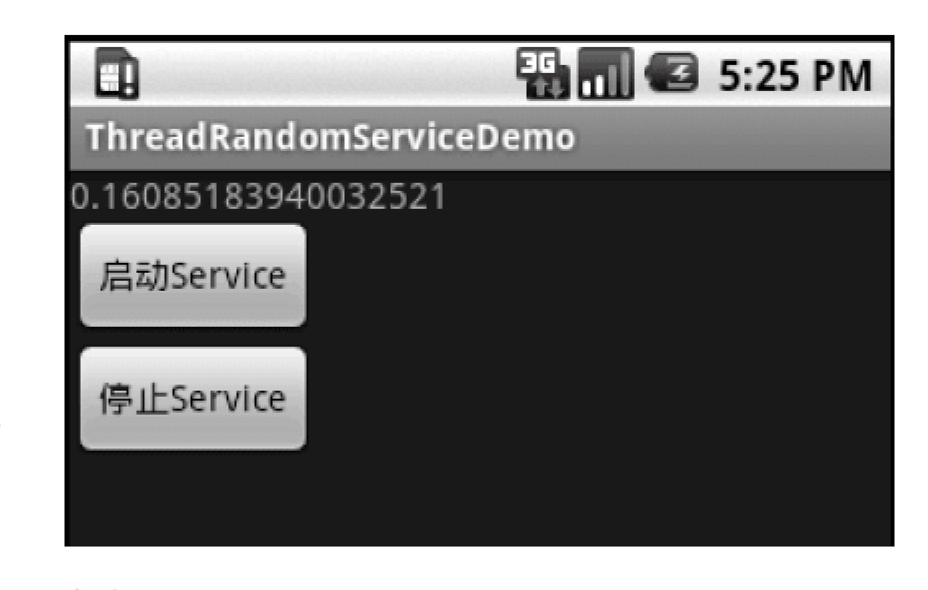



**图 7.4　ThreadRandomServiceDemo 用户界面**

在 ThreadRandomServiceDemo 示例中，RandomService.java 文件是描述 Service 的文件，用来创建线程、产生随机数和调用界面更新函数。ThreadRandomServiceDemo.java 文件是界面的 Activity 文件，

封装 Handler 的界面更新函数就在这个文件中。下面给出 RandomService.java 和 ThreadRandomServiceDemo.java 文件的完整代码。

RandomService.java 文件的完整代码如下：

```
package edu.hrbeu.ThreadRandomServiceDemo;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class RandomService extends Service{

    private Thread workThread;

    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this, "(1) 调用 onCreate()",
                Toast.LENGTH_LONG).show();
        workThread=new Thread(null,backgroudWork,"WorkThread");
    }

    @Override
    public void onStart(Intent intent, int startId) {
          super.onStart(intent, startId);
          Toast.makeText(this, "(2) 调用 onStart()",
                  Toast.LENGTH_SHORT).show();
          if (!workThread.isAlive()){
              workThread.start();
          }
    }

    @Override
    public void onDestroy() {
         super.onDestroy();
         Toast.makeText(this, "(3) 调用 onDestroy()",
                 Toast.LENGTH_SHORT).show();
         workThread.interrupt();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
```

```
        }

        private Runnable backgroudWork=new Runnable(){
            @Override
            public void run() {
                try {
                    while(!Thread.interrupted()){
                        double randomDouble=Math.random();
                        ThreadRandomServiceDemo.UpdateGUI(randomDouble);
                        Thread.sleep(1000);
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
}
```

ThreadRandomServiceDemo.java 文件的完整代码如下：

```
package edu.hrbeu.ThreadRandomServiceDemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class ThreadRandomServiceDemo extends Activity {

        private static Handler handler=new Handler();
        private static TextView labelView=null;
        private static double randomDouble ;

        public static void UpdateGUI(double refreshDouble){
            randomDouble=refreshDouble;
            handler.post(RefreshLable);
        }

        private static Runnable RefreshLable=new Runnable(){
            @Override
            public void run() {
                labelView.setText(String.valueOf(randomDouble));
```

```
            }
        };

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            labelView= (TextView)findViewById(R.id.label);
            Button startButton= (Button)findViewById(R.id.start);
            Button stopButton= (Button)findViewById(R.id.stop);
            final Intent serviceIntent= new Intent(this, RandomService.class);

            startButton.setOnClickListener(new Button.OnClickListener(){
                public void onClick(View view){
                    startService(serviceIntent);
                }
            });

            stopButton.setOnClickListener(new Button.OnClickListener(){
                public void onClick(View view){
                    stopService(serviceIntent);
                }
            });
        }
    }
```

## 7.2.3 服务绑定

以绑定方式使用 Service，能够获取到 Service 对象，不仅能够正常启动 Service，而且能够调用正在运行中的 Service 实现的公有方法和属性。为了使 Service 支持绑定，需要在 Service 类中重载 onBind()方法，并在 onBind()方法中返回 Service 对象，示例代码如下：

```
    public class MathService extends Service{
        private final IBinder mBinder= new LocalBinder();

        public class LocalBinder extends Binder{
            MathService getService() {
                return MathService.this;
            }
        }

        @Override
        public IBinder onBind(Intent intent) {
```

```
12            return mBinder;
13        }
14    }
```

当 Service 被绑定时，系统会调用 onBind()函数，通过 onBind()函数的返回值，将 Service 对象返回给调用者。但从第 11 行代码中可以看出，onBind()函数的返回值必须是符合 IBinder 接口，因此在代码的第 2 行声明一个接口变量 mBinder，mBinder 符合 onBind()函数返回值的要求，因此将 mBinder 传递给调用者。IBinder 是用于进程内部和进程间过程调用的轻量级接口，定义了与远程对象交互的抽象协议，使用时通过继承 Binder 的方法实现。继承 Binder 的代码在第 4 行，LocalBinder 是继承 Binder 的一个内部类，并在代码第 5 行实现了 getService()函数，当调用者获取到 mBinder 后，通过调用 getService()即可获取到 Service 的对象。

调用者通过 bindService()函数绑定服务，并在第 1 个参数中将 Intent 传递给 bindService()函数，声明需要启动的 Service。第 3 个参数 Context.BIND_AUTO_CREATE 表明只要绑定存在，就自动建立 Service；同时也告知 Android 系统，这个 Service 的重要程度与调用者相同，除非考虑终止调用者，否则不要关闭这个 Service。

```
final Intent serviceIntent=new Intent(this,MathService.class);
bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
```

bindService()函数的第 2 个参数是 ServiceConnnection。当绑定成功后，系统将调用 ServiceConnnection 的 onServiceConnected()方法；而当绑定意外断开后，系统将调用 ServiceConnnection 中的 onServiceDisconnected 方法。因此，以绑定方式使用 Service，调用者需要声明一个 ServiceConnnection，并重载内部的 onServiceConnected()方法和 onServiceDisconnected 方法。

```
1    private ServiceConnection mConnection=new ServiceConnection() {
2        @Override
3        public void onServiceConnected(ComponentName name, IBinder service) {
4            mathService=((MathService.LocalBinder)service).getService();
5        }
6        @Override
7        public void onServiceDisconnected(ComponentName name) {
8            mathService=null;
9        }
10   };
```

在代码的第 4 行中，绑定成功后通过 getService()获取 Service 对象，这样便可以调用 Service 中的方法和属性。代码第 8 行将 Service 对象设置为 null，表示绑定意外失效，Service 实例不再可用。

取消绑定仅需要使用 unbindService()方法，并将 ServiceConnnection 传递给 unbindService()方法。但需要注意的是，unbindService()方法成功后，系统并不会调用 onServiceConnected()，因为 onServiceConnected()仅在意外断开绑定时才被调用。

```
unbindService(mConnection);
```

绑定方式中，当调用者通过 bindService()函数绑定 Servcie 时，onCreate()函数和 onBinde()函数将先后被调用。当调用者通过 unbindService()函数取消绑定 Servcie 时，onUnbind()函数将被调用，如果 onUnbind()函数的返回 true，则表示在新调用者绑定服务时，onRebind()函数将被调用。绑定方式的函数调用顺序如图 7.5 所示。

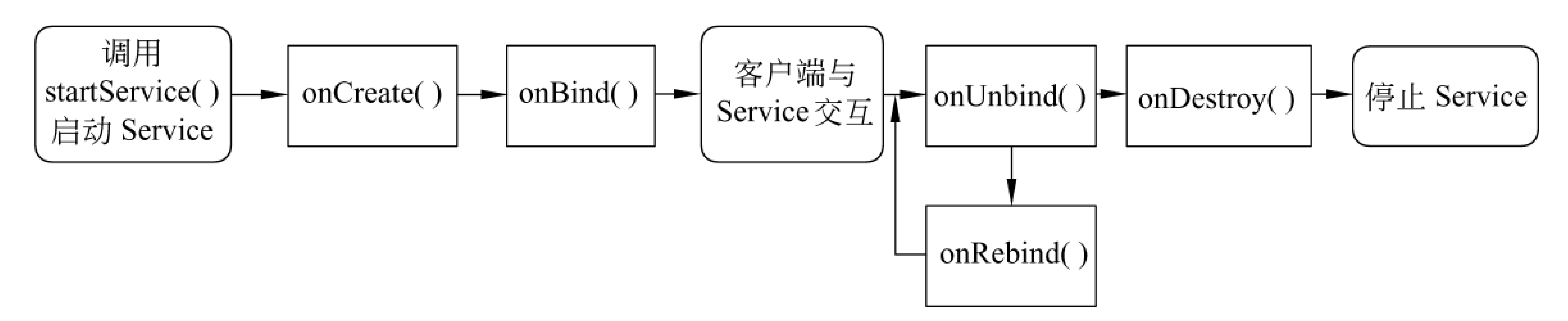



**图 7.5 绑定方式的函数调用顺序**

SimpleMathServiceDemo 是使用绑定方式使用 Service 的示例。在示例中创建了 MathService 服务，用来完成简单的数学运算，这里的数学运算仅包括加法运算，虽然没有实际意义，但足以说明如何使用绑定方式调用 Service 实例中的公有方法。在服务绑定后，用户可以单击"加法运算"按钮，将两个随机产生的数值传递给 MathService 服务，并从 MathService 对象中获取到加法运算的结果，然后显示在屏幕的上方。"取消绑定"按钮可以解除与 MathService 的绑定关系，在取消绑定后，无法通过"加法运算"按钮获取加法运算结果。SimpleMathServiceDemo 的用户界面如图 7.6 所示。




**图 7.6 SimpleMathServiceDemo 用户界面**

在 SimpleMathServiceDemo 示例中，MathService.java 文件是描述 Service 的文件。SimpleMathServiceDemo.java 文件是界面的 Activity 文件，绑定和取消绑定服务的代码在这个文件中。下面给出 MathService.java 和 SimpleMathServiceDemo.java 文件的完整代码。

MathService.java 文件的完整代码如下：

```
package edu.hrbeu.SimpleMathServiceDemo;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

public class MathService extends Service{

```

```
    private final IBinder mBinder= new LocalBinder();

    public class LocalBinder extends Binder{
        MathService getService() {
            return MathService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        Toast.makeText(this, "本地绑定：MathService",
                Toast.LENGTH_SHORT).show();
        return mBinder;
    }

    @Override
    public boolean onUnbind(Intent intent){
        Toast.makeText(this, "取消本地绑定：MathService",
                Toast.LENGTH_SHORT).show();
        return false;
    }


    public long Add(long a, long b){
        return a+b;
    }

}
```

SimpleMathServiceDemo.java 文件的完整代码如下：

```
package edu.hrbeu.SimpleMathServiceDemo;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class SimpleMathServiceDemo extends Activity {
```

```
    private MathService mathService;
    private boolean isBound=false;
    TextView labelView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        labelView=(TextView)findViewById(R.id.label);
        Button bindButton=(Button)findViewById(R.id.bind);
        Button unbindButton=(Button)findViewById(R.id.unbind);
        Button computButton=(Button)findViewById(R.id.compute);

        bindButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(!isBound){
                    final Intent serviceIntent=new Intent(SimpleMath- ServiceDemo.this,
                    MathService.class);
        bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
                    isBound=true;
                }
            }
        });

        unbindButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(isBound){
                    isBound=false;
                    unbindService(mConnection);
                    mathService=null;
                }
            }
        });

        computButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                    if (mathService==null){
                        labelView.setText("未绑定服务");
                        return;
                    }
                    long a=Math.round(Math.random()*100);
```

```
                    long b=Math.round(Math.random() * 100);
                    long result=mathService.Add(a, b);
                    String msg=String.valueOf(a)+"+"+String.valueOf(b)+
                                    "="+String.valueOf(result);
                    labelView.setText(msg);
                }
            });
        }

        private ServiceConnection mConnection=new ServiceConnection() {
            @Override
            public void onServiceConnected(ComponentName name,IBinder service) {
                mathService=((MathService.LocalBinder)service).getService();
            }

            @Override
            public void onServiceDisconnected(ComponentName name) {
                mathService=null;
            }
        };
}
```

# 7.3 跨进程服务

## 7.3.1 进程间通信

在 Android 系统中,每个应用程序在各自的进程中运行,而且出于安全原因的考虑,这些进程之间彼此是隔离的,进程之间传递数据和对象,需要使用 Android 支持的进程间通信(Inter-Process Communication,IPC)机制。在 UNIX/Linux 系统中,传统的 IPC 机制包括共享内存、管道、消息队列和 Socket 等等,这些 IPC 机制虽然被广泛使用,但仍然存在着固有的缺陷,如容易产生错误、难于维护等等。在 Android 系统中,没有使用传统的 IPC 机制,而是采用 Intent 和跨进程服务的方式实现 IPC,使应用程序具有更好的独立性和健壮性。

Android 系统允许应用程序使用 Intent 使用其他 Activity 和 Service,同时 Intent 可以承载数据,是一种极为简单、高效,且易于使用的 IPC 机制。Android 系统的另一种 IPC 机制就是跨进程服务,服务和调用者在不同的两个进程中,调用过程需要跨越进程才能实现。

在 Android 系统中使用跨进程服务,一般按照以下三个步骤实现。首先,使用 AIDL 语言定义跨进程服务的接口。然后根据 AIDL 语言定义的接口,在具体的 Service 类中实现接口中定义的方法和属性。最后在需要调用跨进程服务的组件中,通过相同的

AIDL 接口文件，调用跨进程服务。

## 7.3.2 服务创建与调用

在 Android 系统中，不同进程不能访问其他进程的内存控件，因此为了使数据能够在不同进程间传递，数据必须转换成为能够穿越进程边界的系统级原语，同时，在数据完成进程边界穿越后，还需要转换回原有的格式。AIDL（Android Interface Definition Language）是 Android 系统自定义的接口描述语言，可以简化进程间数据格式转换和数据交换的代码，通过定义 Service 内部的公共方法，允许调用者和 Service 在不同进程间相互传递数据。AIDL 的 IPC 机制与 COM 和 Corba 非常相似，都是基于接口的轻量级进程通信机制。

AIDL 语言的语法与 Java 语言的接口定义非常相似，唯一不同之处，AIDL 允许定义函数参数的传递方向。AIDL 支持三种方向：in、out 和 inout，标识为 in 的参数将从调用者传递到跨进程服务中，标识为 out 的参数将从跨进程服务传递到调用者中，标识为 inout 的参数将先从调用者传递到跨进程服务中，再从跨进程服务返回给调用者。在不标识参数的传递方向时，默认认定所有函数的传递方向为 in。出于性能方面的考虑，不要在参数中标识不需要的传递方向。

远程访问的创建和调用需要使用 AIDL 语言，一般分为以下几个过程：

(1) 使用 AIDL 语言定义跨进程服务的接口。

(2) 通过继承 Service 类实现跨进程服务。

(3) 绑定和使用跨进程服务。

下面以 RemoteMathServiceDemo 示例为参考，说明如何创建跨进程服务。在这个示例中，仅定义了 MathService 服务，可以为远程调用者提供加法服务。

### 1. 使用 AIDL 语言定义跨进程服务的接口

首先使用 AIDL 语言定义的 MathService 的服务接口，文件名为 IMathService.aidl。

```
package edu.hrbeu.RemoteMathServiceDemo;
interface IMathService {
    long Add(long a, long b);
}
```

从上面的代码中可以看出，IMathService 接口仅包含一个 add()方法，传入的参数是两个长型整数，返回值也是长型整数。

如果使用 Eclipse 编辑 IMathService.aidl 文件，当用户保存文件后，ADT 会自动在/gen 目录下生成 IMathService.java 文件，文件结构如图 7.7 所示。

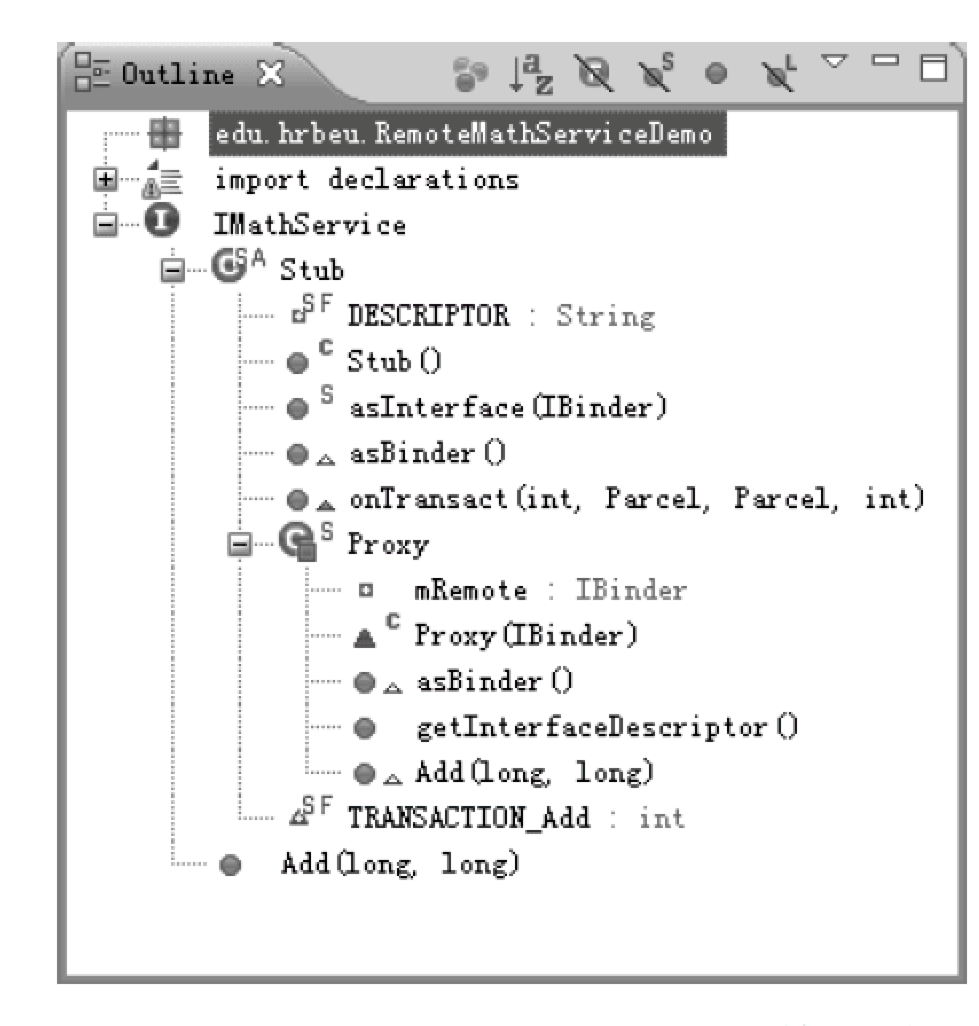



图 7.7 IMathService.java 文件结构

IMathService.java 文件根据 IMathService.aidl 的定义，生成了一个内部静态抽象类 Stub，Stub 继承了 Binder 类，并实现 ImathService 接口。在 Stub 类中，还包含一个重要的静态类 Proxy。如果认为 Stub 类实现进程内服务调用，那么 Proxy 类则是用来实现跨进程服务调用的，将 Proxy 作为 Stub 的内部类完全是出于使用方便的目的。Stub 类和 Proxy 类关系如图 7.8 所示。

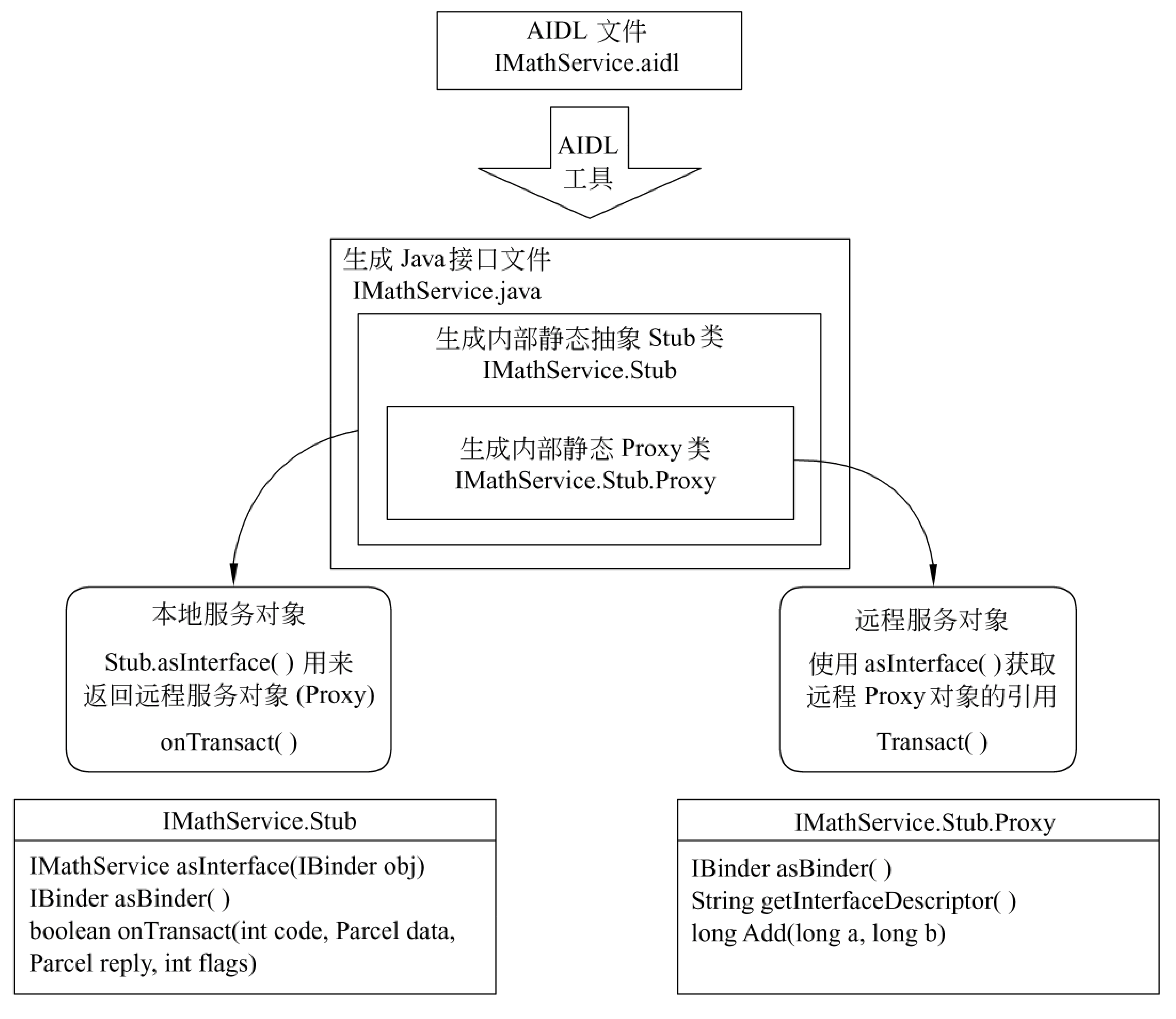



**图 7.8　Stub 类和 Proxy 类关系图**

下面给出 IMathService.java 的完成代码：

```
package edu.hrbeu.RemoteMathServiceDemo;
import java.lang.String;
import android.os.RemoteException;
import android.os.IBinder;
import android.os.IInterface;
import android.os.Binder;
import android.os.Parcel;
public interface IMathService extends android.os.IInterface{
/* * Local-side IPC implementation stub class. * /
public static abstract class Stub extends android. os. Binder implements edu. hrbeu.
RemoteMathServiceDemo.IMathService{
```

```
        private static final java. lang. String DESCRIPTOR = " edu. hrbeu. RemoteMathServiceDemo.
        IMathService";
    /* * Construct the stub at attach it to the interface. * /
        public Stub(){
            this.attachInterface(this, DESCRIPTOR);
        }
    /* *
      * Cast an IBinder object into an IMathService interface,
      * generating a proxy if needed.
      * /
        public static edu.hrbeu.RemoteMathServiceDemo.IMathService asInterface (android.os.IBinder
        obj){
        if ((obj==null)) {
            return null;
        }
        android.os.IInterface iin= (android.os.IInterface)
        obj.queryLocalInterface(DESCRIPTOR);
    if (((iin!=null)&&(iin instanceof edu.hrbeu.RemoteMathServiceDemo.IMathService))) {
        return ((edu.hrbeu.RemoteMathServiceDemo.IMathService)iin);
        }
        return new edu.hrbeu.RemoteMathServiceDemo.IMathService.Stub.Proxy(obj);
    }
    public android.os.IBinder asBinder(){
        return this;
    }
    public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel reply, int flags)
    throws android.os.RemoteException{
        switch (code)    {
            case INTERFACE_TRANSACTION:
            {
                reply.writeString(DESCRIPTOR);
                return true;
            }
            case TRANSACTION_Add:
            {
                data.enforceInterface(DESCRIPTOR);
                long _arg0;
                _arg0=data.readLong();
                long _arg1;
                _arg1=data.readLong();
                long _result=this.Add(_arg0, _arg1);
                reply.writeNoException();
                reply.writeLong(_result);
                return true;
```

```
51              }
52          }
53          return super.onTransact(code, data, reply, flags);
54      }
55      private static class Proxy implements edu.hrbeu.RemoteMathServiceDemo.IMathService{
56          private android.os.IBinder mRemote;
57          Proxy(android.os.IBinder remote){
58              mRemote= remote;
59          }
60          public android.os.IBinder asBinder(){
61              return mRemote;
62          }
63          public java.lang.String getInterfaceDescriptor(){
64              return DESCRIPTOR;
65          }
66          public long Add(long a, long b) throws android.os.RemoteException{
67              android.os.Parcel _data= android.os.Parcel.obtain();
68              android.os.Parcel _reply= android.os.Parcel.obtain();
69              long _result;
70              try {
71                  _data.writeInterfaceToken(DESCRIPTOR);
72                  _data.writeLong(a);
73                  _data.writeLong(b);
74                  mRemote.transact(Stub.TRANSACTION_Add, _data, _reply, 0);
75                  _reply.readException();
76                  _result= _reply.readLong();
77              }
78              finally {
79                  _reply.recycle();
80                  _data.recycle();
81              }
82          return _result;
83          }
84      }
85          static final int TRANSACTION_Add= (IBinder.FIRST_CALL_TRANSACTION+ 0);
86      }
87      public long Add(long a, long b) throws android.os.RemoteException;
88      }
```

IMathService 继承了 android. os. IInterface(第 8 行),这是所有使用 AIDL 建立的接口都必须继承的基类接口,这个基类接口中定义了 asBinder()方法,用来获取 Binder 对象。在代码的第 30 行到第 32 行,实现了 android. os. IInterface 接口所定义的 asBinder()方法。在 IMathService 中,绝大多数的代码是用来实现 Stub 这个抽象类的。每个远程接口都包 Stub 类,因为是内部类,所有并不会产生命名的冲突。

asInterface(IBinder)是 Stub 内部的跨进程服务接口,调用者可以通过该方法获取到跨进程服务的对象。仔细观察 asInterface(IBinder)实现方法,首先判断 IBinder 对象 obj 是否为 null(第 21 行),如果是则立即返回。然后使用 DESCRIPTOR 构造 android. os. IInterface 对象(第 24 行),并判断 android. os. IInterface 对象是否为进程内服务,如果是进程内服务,则无需进程间通信,返回 android. os. IInterface 对象(第 26 行);如果不是进程内服务,则构造并返回 Proxy 对象(第 28 行)。

Proxy 内部包含与 IMathService. aidl 相同签名的函数(第 66 行),并且在该函数中以一定的顺序将所有参数写入 Parcel 对象(第 71～76 行),以供 Stub 内部的 onTransact()方法能够正确获取到参数。

当数据以 Parcel 对象的形式传递到跨进程服务的内部时,onTransact()方法(第 33 行)将从 Parcel 对象中逐一的读取每个参数,然后调用 Service 内部制定的方法,并再将结果写入另一个 Parcel 对象,准备将这个 Parcel 对象返回给远程的调用者。

Parcel 是 Android 系统中应用程序进程间数据传递的容器,能够在两个进程中完成数据的打包和拆包的工作,但 Parcel 不同于通用意义上的序列化,Parcel 的设计目的是用于高性能 IPC 传输,因此不能将 Parcel 对象保存在任何持久存储设备上。

**2. 通过继承 Service 类实现跨进程服务**

IMathService. aidl 是对跨进程服务接口的定义,自动生成的 IMathService. java 内部实现了跨进程服务数据传递的相关方法,下一步介绍如何实现跨进程服务。实现跨进程服务需要建立一个继承 android. app. Service 的类,并在该类中通过 onBind()方法返回 IBinder 对象,调用者使用返回的 IBinder 对象就可访问跨进程服务。IBinder 对象的建立通过使用 IMathService. java 内部的 Stub 类实现,并逐一实现在 IMathService. aidl 接口文件定义的函数。在 RemoteMathServiceDemo 示例中,跨进程服务的实现类是 MathService. java,下面是 MathService. java 的完整代码:

```
package edu.hrbeu.RemoteMathServiceDemo;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MathService extends Service{
    private final IMathService.Stub mBinder= new IMathService.Stub() {
        public long Add(long a, long b) {
            return a+b;
        }
    };
    @Override
    public IBinder onBind(Intent intent) {
        Toast.makeText(this, "远程绑定:MathService",
```

```
17                     Toast.LENGTH_SHORT).show();
18             return mBinder;
19         }
20         @Override
21         public boolean  onUnbind  (Intent intent){
22             Toast.makeText(this, "取消远程绑定：MathService",
23                     Toast.LENGTH_SHORT).show();
24             return false;
25         }
26     }
```

第 8 行代码表明 MathService 继承于 android. app. Service。第 9 行建立 IMathService. Stub 的对象 mBinder，并在第 10 行实现了 AIDL 文件定义的跨进程服务接口。第 18 行在 onBind()方法中，将 mBinder 返回给远程调用者。第 16 行和第 22 行分别是在绑定和取消绑定时，为用户产生的提示信息。

RemoteMathServiceDemo 示例的文件结构如图 7.9 所示。示例中只有跨进程服务的类文件 MathService. java 和接口文件 IMathService. aidl，没有任何用于启动时显示用户界面的 Activity 文件。

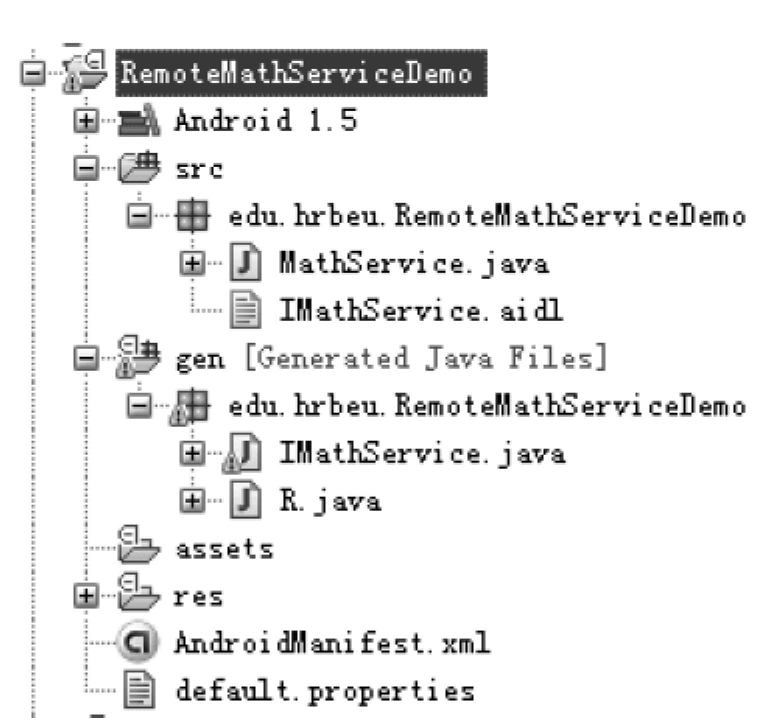



**图 7.9　RemoteMathServiceDemo 文件结构**

在调试 RemoteMathServiceDemo 示例时，模拟器的屏幕上不会出现用户界面，但在控制台会有“没有找到用于启动的 Activity，仅将应用程序同步到设备上”的提示信息，如图 7.10 所示。这些信息表明 apk 文件已经上传到模拟器中。

```
Android Launch!
adb is running normally.
No Launcher activity found!
The launch will only sync the application package on the device!
Performing sync
Automatic Target Mode: using existing emulator 'emulator-5554' running compatible
Uploading RemoteMathServiceDemo.apk onto device 'emulator-5554'
Installing RemoteMathServiceDemo.apk...
Application already exists. Attempting to re-install instead...
Success!
\RemoteMathServiceDemo\bin\RemoteMathServiceDemo.apk installed on device
Done!
```

**图 7.10　RemoteMathServiceDemo 调试信息**

为了进一步确认编译好的 apk 文件是否正确上传到模拟器中，可以使用 File Explorer 查看模拟器的文件系统。如果能在/data/app/下找到 edu. hrbeu. RemoteMathServiceDemo. apk 文件，说明提供跨进程服务的 apk 文件已经正确上传。RemoteMathServiceDemo 示例无法在 Android 模拟器的程序启动栏中找到，只能通过其他应用程序调用该示例中的跨进程服务。图 7.11 表明了 edu. hrbeu. Remote-MathServiceDemo. apk 文件的保存位置。

| Name | Size | Date | Time | Permiss... |
|---|---|---|---|---|
| ⊟ data | | 2009-06-20 | 00:53 | drwxrwx--x |
| ⊞ anr | | 2009-06-20 | 00:54 | drwxrwxrwx |
| ⊟ app | | 2009-06-20 | 00:53 | drwxrwx--x |
| edu.hrbeu.RemoteMathServiceDemo.apk | 9846 | 2009-07-07 | 01:37 | -rw-r--r-- |
| edu.hrbeu.SimpleMathServiceDemo.apk | 10920 | 2009-07-05 | 09:31 | -rw-r--r-- |
| edu.hrbeu.SimpleRandomServiceDemo.apk | 10004 | 2009-07-07 | 02:14 | -rw-r--r-- |
| edu.hrbeu.ThreadRandomServiceDemo.apk | 10686 | 2009-07-07 | 02:14 | -rw-r--r-- |
| ⊞ app-private | | 2009-06-20 | 00:53 | drwxrwx--x |
| ⊞ dalvik-cache | | 2009-06-20 | 00:53 | drwxrwx--x |
| ⊞ data | | 2009-06-20 | 00:53 | drwxrwx--x |
| ⊞ local | | 2009-06-20 | 00:53 | drwxrwx--x |
| ⊞ lost+found | | 2009-06-20 | 00:53 | drwxrwx--- |
| ⊞ misc | | 2009-06-20 | 00:53 | drwxrwx--t |
| ⊞ property | | 2009-06-20 | 00:53 | drwx------ |
| ⊞ system | | 2009-06-20 | 00:54 | drwxrwxr-x |
| ⊞ sdcard | | 2009-07-07 | 01:35 | d--------- |
| ⊞ system | | 2009-04-22 | 04:11 | drwxr-xr-x |

**图 7.11　RemoteMathServiceDemo.apk 文件位置**

RemoteMathServiceDemo 是本书中第一个没有 Activity 的示例，在 AndroidManifest.xml 文件中，在<application>标签下只有一个<service>标签。

AndroidManifest.xml 文件的完整代码如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="edu.hrbeu.RemoteMathServiceDemo"
4       android:versionCode="1"
5       android:versionName="1.0">
6       <application android:icon="@drawable/icon" android:label="@string/app_name">
7           <service android:name=".MathService"
8               android:process=":remote">
9               <intent-filter>
10                  <action android:name="edu.hrbeu.RemoteMathServiceDemo.MathService" />
11              </intent-filter>
12          </service>
13      </application>
14      <uses-sdk android:minSdkVersion="3" />
15  </manifest>
```

注意第 10 行代码，edu.hrbeu.RemoteMathServiceDemo.MathService 是远程调用 MathService 的标识，在调用者段使用 Intent.setAction()函数将标识加入 Intent 中，然后隐式启动或绑定服务。

### 3. 绑定和使用跨进程服务

RemoteMathCallerDemo 示例说明如何调用 RemoteMathServiceDemo 示例中的跨进程服务。RemoteMathCallerDemo 的界面如图 7.12 所示，用户可以绑定跨进程服务，也可以取消服务绑定。在绑定跨进程服务后，可以调用 RemoteMathServiceDemo 中的 MathService 服务进行加法运算，运算的输入由 RemoteMathCallerDemo 随机产生，运算

的输入和结果显示在屏幕的上方。



图 7.12 RemoteMathCallerDemo 用户界面

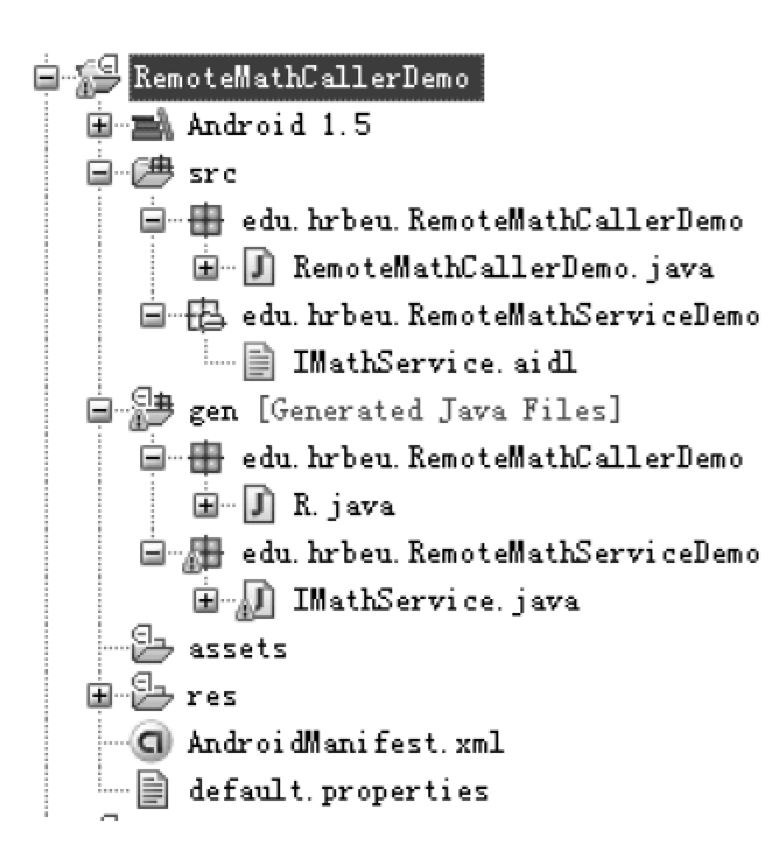


图 7.13 RemoteMathCallerDemo 的文件结构

应用程序在调用跨进程服务时,应用程序与跨进程服务应具有相同的 Proxy 类和签名函数,这样才能使数据在调用者处打包后,可以在远程访问端正确拆包,反之亦然。从实践角度来讲,调用者需要使用与跨进程服务端相同的 AIDL 文件。在 RemoteMathCallerDemo 示例,在 edu.hrbeu.RemoteMathServiceDemo 包下,引入与 RemoteMathServiceDemo 相同的 AIDL 文件 IMathService.aidl,同时在/gen 目录下会自动产生相同的 IMathService.java 文件。RemoteMathServiceDemo 的文件结构如图 7.13 所示。

RemoteMathCallerDemo.java 是 Activity 的文件,跨进程服务的绑定和使用方法与 7.2.3 节的进程内服务绑定示例 SimpleMathServiceDemo 相似。不同之处主要包括以下两个方面,一是使用 IMathService 声明跨进程服务对象(代码第 1 行);二是通过 IMathService.Stub 的 asInterface()方法实现获取服务对象(代码第 6 行)。

```
1	private IMathService mathService;
2
3	private ServiceConnection mConnection = new ServiceConnection() {
4		@Override
5		public void onServiceConnected(ComponentName name, IBinder service) {
6			mathService = IMathService.Stub.asInterface(service);
7		}
8		@Override
9		public void onServiceDisconnected(ComponentName name) {
10			mathService = null;
11		}
12	};
```

绑定服务时,首先通过 setAction()方法声明服务标识,然后调用 bindService()绑定服务。服务标识必须与跨进程服务在 AndroidManifest.xml 文件中声明的服务标识完全相同。因此本示例的服务标识为 edu.hrbeu.RemoteMathServiceDemo.MathService,与跨进程服务示例 RemoteMathServiceDemo 在 AndroidManifest.xml 文件声明的服务标

识一致。

```
final Intent serviceIntent=new Intent();
serviceIntent.setAction("edu.hrbeu.RemoteMathServiceDemo.MathService");
bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
```

下面给出 RemoteMathCallerDemo.java 文件的完整代码：

```
package edu.hrbeu.RemoteMathCallerDemo;

import edu.hrbeu.RemoteMathServiceDemo.IMathService;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class RemoteMathCallerDemo extends Activity {
    private IMathService mathService;

    private ServiceConnection mConnection=new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name,IBinder service) {
            mathService=IMathService.Stub.asInterface(service);
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            mathService=null;
        }
    };

    private boolean isBound=false;
    TextView labelView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

```

```
        labelView= (TextView)findViewById(R.id.label);
        Button bindButton= (Button)findViewById(R.id.bind);
        Button unbindButton= (Button)findViewById(R.id.unbind);
        Button computButton= (Button)findViewById(R.id.compute_add);

        bindButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(!isBound){
                    final Intent serviceIntent= new Intent();
                    serviceIntent.setAction("edu.hrbeu.RemoteMathServiceDemo.MathService");
                    bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
                    isBound= true;
                }
            }
        });

        unbindButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(isBound){
                    isBound= false;
                    unbindService(mConnection);
                    mathService= null;
                }
            }
        });

        computButton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                    if (mathService==null){
                        labelView.setText("未绑定跨进程服务");
                        return;
                    }
                    long a=Math.round(Math.random() * 100);
                    long b=Math.round(Math.random() * 100);
                    long result= 0;
                    try {
                        result=mathService.Add(a, b);
                    } catch (RemoteException e) {
                        e.printStackTrace();
                    }
```

```
                        String msg= String.valueOf(a)+ "+ "+ String.valueOf(b)+
                                    "= "+ String.valueOf(result);
                        labelView.setText(msg);
                  }
            });
      }
}
```

## 7.3.3 数据传递

在 Android 系统中，进程间传递的数据包括 Java 语言支持的基本数据类型和用户自定义的数据类型，为了使数据能够穿越进程边界，所有数据都必须是“可打包”的。对于 Java 语言的基本数据类型，打包过程是自动完成的。但对于自定义的数据类型，用户则需要实现 Parcelable 接口，使自定义的数据类型能够转换为系统级原语保存在 Parcel 对象中，穿越进程边界后可再转换为初始格式。AIDL 支持的数据类型见表 7.1。

**表 7.1 AIDL 支持的数据类型**

| 类　型 | 说　明 | 需要引入 |
|---|---|---|
| Java 语言的基本类型 | 包括 boolean、byte、short、int、float 和 double 等 | 否 |
| String | java. lang. String | 否 |
| CharSequence | java. lang. CharSequence | 否 |
| List | 其中所有的元素都必须是 AIDL 支持的数据类型 | 否 |
| Map | 其中所有的键和元素都必须是 AIDL 支持的数据类型 | 否 |
| 其他 AIDL 接口 | 任何其他使用 AIDL 语言生成的接口类型 | 是 |
| Parcelable 对象 | 实现 Parcelable 接口的对象 | 是 |

下面以 ParcelMathServiceDemo 示例为参考，说明如何在跨进程服务中使用自定义数据类型。这个示例是 RemoteMathServiceDemo 示例的延续，也定义了 MathService 服务，同样可以为远程调用者提供加法服务。而且同样也是没有启动界面，因此在模拟器的调试过程与 RemoteMathServiceDemo 示例相同。

不同之处在于 MathService 服务增加了“全运算”功能，在接收到输入参数后，将向调用者返回一个包含“加、减、乘、除”全部运算结果的对象。这个对象是一个自定义的类，为了能够使其他 AIDL 文件可使用这个自定义类，需要使用 AIDL 语言声明这个类。

首先建立 AllResult. aidl 文件，声明 AllResult 类。在第 2 行代码中使用 Parcelable 声明自定义类，这样其他的 AIDL 文件就可以使用这个自定义的类。ParcelMathServiceDemo 的文件结构如图 7.14 所示。

```
1    package edu.hrbeu.ParcelMathServiceDemo;
2    parcelable AllResult;
```

在 IMathService.aidl 文件中，代码第 6 行为全运算增加了新的函数 ComputeAll()，该函数的返回值就是在 AllResult.aidl 文件中定义的 AllResult。同时，为了能够使用自定义数据结构 AllResult，务必在代码中引入 edu.hrbeu.ParcelMathServiceDemo.AllResult。第 2 行和第 6 行是新增的代码，其他的代码与 RemoteMathServiceDemo 示例相同。

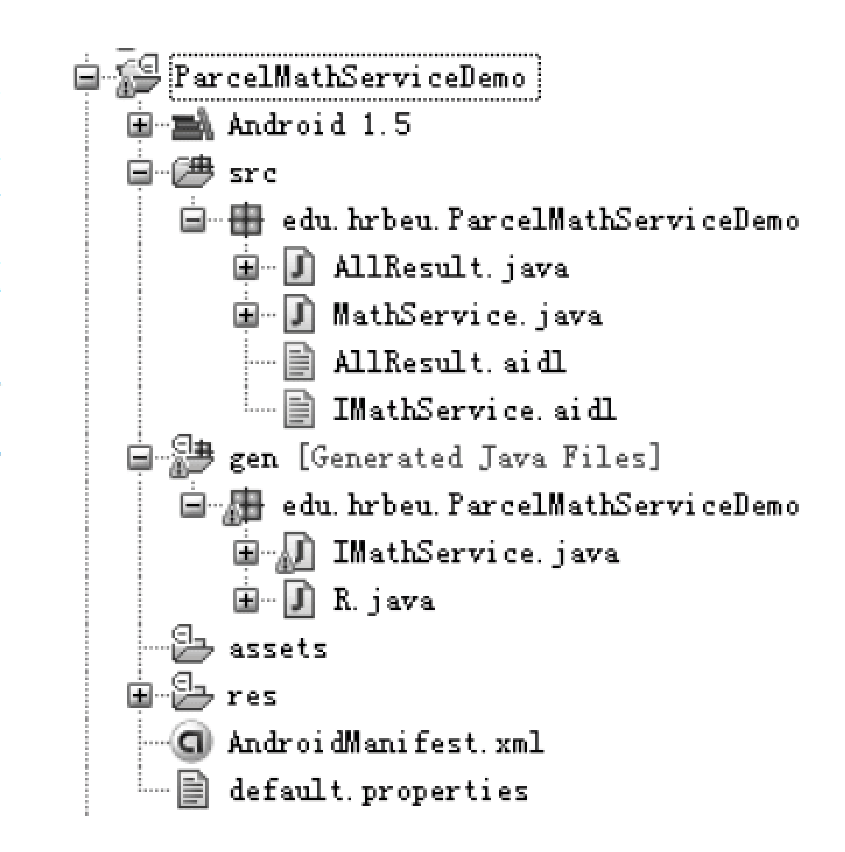



**图 7.14 ParcelMathServiceDemo 的文件结构**

```
1    package edu.hrbeu.ParcelMathServiceDemo;
2    import edu.hrbeu.ParcelMathServiceDemo.AllResult;
3
4    interface IMathService {
5        long Add(long a, long b);
6        AllResult ComputeAll(long a, long b);
7    }
```

在 AIDL 文件定义完毕后，下一步介绍如何构造 AllResult 类。AllResult 类除了基本的构造函数以外，还需要有以 Parcel 对象为输入的构造函数，并且需要重载打包函数 writeToParcel()。AllResult.java 文件的完整代码如下：

```
package edu.hrbeu.ParcelMathServiceDemo;

import android.os.Parcel;
import android.os.Parcelable;

public class AllResult implements Parcelable {
    public long AddResult;
    public long SubResult;
    public long MulResult;
    public double DivResult;

    public AllResult(long addRusult, long subResult, long mulResult, double divResult){
        AddResult= addRusult;
        SubResult= subResult;
        MulResult=mulResult;
        DivResult= divResult;
    }

    public AllResult(Parcel parcel) {
        AddResult=parcel.readLong();
        SubResult=parcel.readLong();
        MulResult=parcel.readLong();
        DivResult=parcel.readDouble();
    }
```

```
25
26        @Override
27        public int describeContents() {
28            return 0;
29        }
30
31        @Override
32        public void writeToParcel(Parcel dest, int flags) {
33            dest.writeLong(AddResult);
34            dest.writeLong(SubResult);
35            dest.writeLong(MulResult);
36            dest.writeDouble(DivResult);
37        }
38
39        public static final Parcelable.Creator<AllResult> CREATOR=
40            new Parcelable.Creator<AllResult>(){
41            public AllResult createFromParcel(Parcel parcel){
42                return new AllResult(parcel);
43            }
44            public AllResult[] newArray(int size){
45                return new AllResult[size];
46            }
47        };
48    }
```

代码第 6 行说明了 AllResult 类继承于 Parcelable。代码第 7 行到第 10 行用来保存全运算的运算结果。第 12 行是 AllResult 类的基本构造函数。第 19 行也是类的构造函数，支持通过 Parcel 对象实例化 AllResult。代码第 32 行的 writeToParcel()是“打包”函数，将 AllResult 类内部的数据，按照特定的顺序写入 Parcel 对象，写入的顺序必须与构造函数的读取顺序一致（代码第 20 行到第 23 行）。第 39 行实现了静态公共字段 Creator，用来使用 Parcel 对象构造 AllResult 对象。

在 MathService.java 文件中，增加了用来进行全运算的 ComputAll()函数，并将运算结果保存在 AllResult 对象中。ComputAll()函数实现代码如下：

```
@Override
public AllResult ComputeAll(long a, long b) throws RemoteException {
    long addRusult=a+b;
    long subResult=a-b;
    long mulResult=a * b;
    double divResult= (double) a / (double)b;
    AllResult allResult=new AllResult(addRusult, subResult, mulResult, divResult);
    return allResult;
}
```

ParcelMathCallerDemo 示例是 ParcelMathServiceDemo 示例中 MathService 服务的调用者，文件结构如图 7.15 所示。其中，AllResult.aidl、AllResult.java 和 IMathService.aidl 文件务必与 ParcelMathServiceDemo 示例的三个文件完全一致，否则会出现错误。

在图 7.16 的 ParcelMathCallerDemo 用户界面中可以看出，原来的“加法运算”按钮改为了“全运算”按钮，运算结果显示在界面的下方。

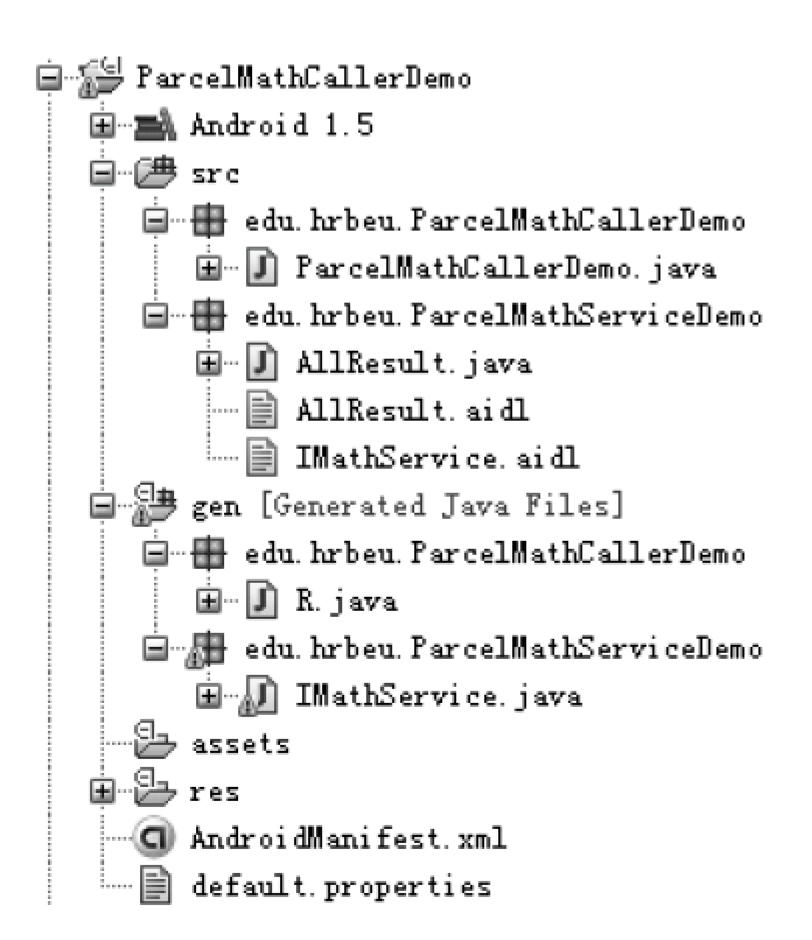



图 7.15　ParcelMathCallerDemo 的文件结构




图 7.16　ParcelMathCallerDemo 用户界面

下面也仅给出 ParcelMathCallerDemo.java 文件与 RemoteMathCallerDemo 示例中 RemoteMathCallerDemo.java 文件不同的一段代码。定义了“全运算”按钮的监听函数，随机产生输入值，调用跨进程服务，获取运算结果，并将运算结果显示在用户界面上。

```
computAllButton.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        if (mathService==null){
            labelView.setText("未绑定跨进程服务");
            return;
        }
        long a=Math.round(Math.random() * 100);
        long b=Math.round(Math.random() * 100);
        AllResult result=null;
        try {
            result=mathService.ComputeAll(a, b);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        String msg="";
        if (result !=null){
            msg+=String.valueOf(a)+"+"+String.valueOf(b)+"="+String.valueOf(result.
            AddResult)+"\n";
            msg+=String.valueOf(a)+"-"+String.valueOf(b)+"="+String.valueOf(result.
            SubResult)+"\n";
```

```
                msg+=String.valueOf(a)+" * "+String.valueOf(b)+"="+String.valueOf(result.MulResult)+"\n";
                msg+=String.valueOf(a)+" / "+String.valueOf(b)+"="+String.valueOf(result.DivResult);
            }
            labelView.setText(msg);
        }
    });
```

## 习　　题

1. 简述 Service 的基本原理和用途。

2. 编程建立一个简单的进程内服务，实现比较两个整数大小的功能。服务提供 Int Compare(Int，Int)函数，输入两个整数，输出较大的整数。

3. 使用 AIDL 语言实现功能与第 2 题相同的跨进程服务。

# 第 8 章 chapter 8

# 数据存储与访问

Android 平台提供多种数据存储方法，包括易于使用的 SharedPreferences、经典的文件存储和轻量级的 SQLite 数据库。通过本章的学习，读者可以了解 Android 平台各种组件数据存储方法的特点和使用方法，掌握跨进程的数据共享方法。

**本章学习目标：**

- 掌握 SharedPreferences 的使用方法
- 掌握各种文件存储的区别与适用情况
- 了解 SQLite 数据库的特点和体系结构
- 掌握 SQLite 数据库的建立和操作方法
- 理解 ContentProvider 的用途和原理
- 掌握 ContentProvider 的创建与使用方法

## 8.1 简单存储

### 8.1.1 SharedPreferences

在应用程序的使用过程中，用户经常会根据自己的习惯，更改应用程序的设置，或设定个性化的内容。为了能够保存用户的设置和个性化内容，应用程序一般在文件系统中保存一个配置文件，并在每次应用程序启动时，读取这个配置文件中的内容。在文件系统中使用配置文件，需要注意配置文件的格式，一般使用 INI 文件或 XML 文件，当然也可以自定义文件格式。INI 文件格式简单，容易读懂，但需要写代码实现文件读取和写入。XML 文件有成熟的类支持，在代码方面更容易实现，但可读性上要比 INI 文件差一些。无论是使用 INI 文件，还是使用 XML 文件保存用户的配置和个性化内容，程序的开发人员是要进行繁琐的编码实现的。

Android 为开发人员提供了更为简单的数据存储方法 SharedPreferences。这是一种轻量级的数据保存方式，通过 SharedPreferences 开发人员可以将 NVP（Name/Value Pair，名称/值对）保存在 Android 的文件系统中，而且 SharedPreferences 完全屏蔽对文件系统的操作过程，开发人员仅是通过调用 SharedPreferences 对 NVP 进行保存和读取。

SharedPreferences不仅能够保存数据，还能够实现不同应用程序间的数据共享。SharedPreferences支持三种访问模式：私有（MODE_PRIVATE）、全局读（MODE_WORLD_READABLE）和全局写（MODE_WORLD_WRITEABLE）。如果将SharedPreferences定义为私有模式，仅有创建程序有权限对其进行读取或写入；如果将SharedPreferences定义为全局读模式，不仅创建程序可以对其进行读取或写入，其他应用程序也读取操作的权限，但没有写入操作的权限；如果将SharedPreferences定义为全局写模式，则创建程序和其他程序都可以对其进行写入操作，但没有读取的权限。

在使用SharedPreferences前，先定义SharedPreferences的访问模式。下面的代码将访问模式定义为私有模式。

```
public static int MODE=MODE_PRIVATE;
```

有的时候需要将SharedPreferences的访问模式设定为既可以全局读，也可以全局写，这样就需要将两种模式写成下面的方式。

```
public static int MODE=Context.MODE_WORLD_READABLE+Context.MODE_WORLD_WRITEABLE;
```

除了定义SharedPreferences的访问模式，还要定义SharedPreferences的名称，这个名称与在Android文件系统中保存的文件同名。因此，只要具有相同的SharedPreferences名称的NVP内容，都会保存在同一个文件中。

```
public static final String PREFERENCE_NAME="SaveSetting";
```

为了可以使用SharedPreferences，需要将访问模式和SharedPreferences名称作为参数，传递到getSharedPreferences()函数，并获取到SharedPreferences对象。

```
SharedPreferences sharedPreferences=getSharedPreferences(PREFERENCE_NAME, MODE);
```

在获取到SharedPreferences对象后，则可以通过SharedPreferences.Editor类对SharedPreferences进行修改，最后调用commit()函数保存修改内容。SharedPreferences广泛支持各种基本数据类型，包括整型、布尔型、浮点型和长型等等。

```
SharedPreferences.Editor editor=sharedPreferences.edit();
editor.putString("Name", "Tom");
editor.putInt("Age", 20);
editor.putFloat("Height", 1.81f);
editor.commit();
```

如果需要从已经保存的SharedPreferences中读取数据，同样是调用getSharedPreferences()函数，并在函数的第1个参数中指明需要访问的SharedPreferences名称，最后通过get＜Type＞()函数获取保存在SharedPreferences中的NVP。get＜Type＞()函数的第1个参数是NVP的名称，第2个参数是在无法获取到数值的时候使用的默认值。

```
SharedPreferences sharedPreferences=getSharedPreferences(PREFERENCE_NAME, MODE);
String name=sharedPreferences.getString("Name","Default Name");
```

```
int age= sharedPreferences.getInt("Age", 20);
float height= sharedPreferences.getFloat("Height",1.81f);
```

## 8.1.2 示例

在介绍了 SharedPreferences 的使用方法后，下面将通过 SimplePreferenceDemo 示例介绍具体说明 SharedPreferences 的文件保存位置和保存格式。SimplePreferenceDemo 示例的用户界面如图 8.1 所示，用户在界面上的输入的信息，将通过 SharedPreferences 在 Activity 关闭时进行保存。当应用程序重新开启时，保存在 SharedPreferences 的信息将被读取出来，并重新呈现在用户界面上。

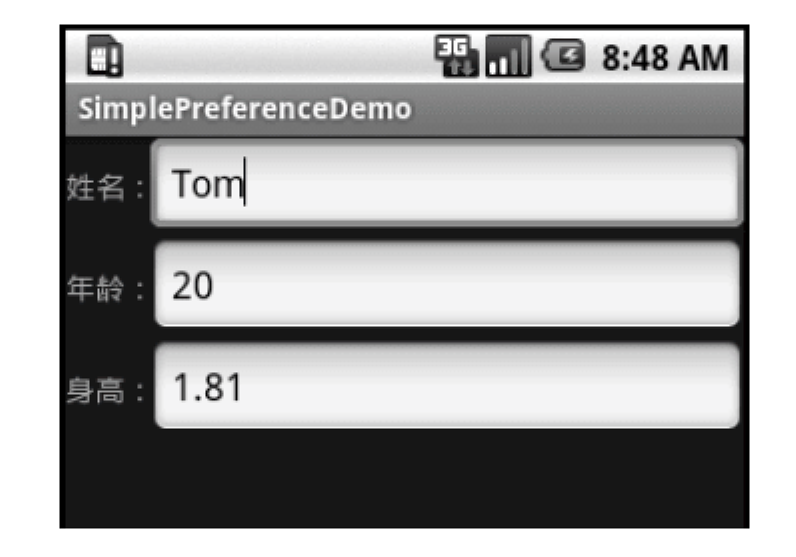



**图 8.1 SimplePreferenceDemo 用户界面**

SimplePreferenceDemo 示例运行后，通过 FileExplorer 查看/data/data 下的数据，Android 为每个应用程序建立了与包同名的目录，用来保存应用程序产生的数据，这些数据包括文件、SharedPreferences 文件和数据库等。SharedPreferences 文件就保存在/data/data/<package name>/shared_prefs 目录下。

在本示例中，shared_prefs 目录下生成了一个名为 SaveSetting.xml 的文件，如图 8.2 所示。这个文件就是保存 SharedPreferences 的文件，文件大小为 170 字节，在 Linux 下的权限为“-rw-rw-rw”。在 Linux 系统中，文件权限分别描述了创建者、同组用户和其他用户对文件的操作限制。x 表示可执行，r 表示可读，w 表示可写，d 表示目录，-表示普通文件。因此，“-rw-rw-rw”表示 SaveSetting.xml 可以被创建者、同组用户和其他用户进行读取和写入操作，但不可执行。产生这样的文件权限与程序人员设定的 SharedPreferences 的访问模式有关，“-rw-rw-rw”的权限是“全局读+全局写”的结果。如果将 SharedPreferences 的访问模式设置为私有，则文件权限将成为“-rw-rw ---”，表示仅有创建者和同组用户具有读写文件的权限。

| | | | | |
|---|---|---|---|---|
| edu.hrbeu.SimplePreferenceDemo | | 2009-07-10 | 02:18 | drwxr-xr-x |
| lib | | 2009-07-10 | 02:18 | drwxr-xr-x |
| shared_prefs | | 2009-07-10 | 03:01 | drwxrwx--x |
| SaveSetting.xml | 170 | 2009-07-15 | 08:45 | -rw-rw-rw- |
| edu.hrbeu.SimpleRandomServiceDemo | | 2009-06-30 | 12:17 | drwxr-xr-x |
| edu.hrbeu.SpinnerDemo | | 2009-06-21 | 07:01 | drwxr-xr-x |

**图 8.2 SaveSetting.xml 文件**

SaveSetting.xml 文件是以 XML 格式保存的信息，内容如下：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <float name="Height" value="1.81" />
    <string name="Name">Tom</string>
    <int name="Age" value="20" />
</map>
```

SimplePreferenceDemo 示例在 onStart()函数中调用 loadSharedPreferences()函数，读取保存在 SharedPreferences 中的姓名、年龄和身高信息，并显示在用户界面上。当 Activity 关闭时，在 onStop()函数调用 saveSharedPreferences()，保存界面上的信息。下面给出示例的完整代码。

SimplePreferenceDemo. java 的完整代码如下：

```
package edu.hrbeu.SimplePreferenceDemo;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.EditText;

public class SimplePreferenceDemo extends Activity {

    private EditText nameText;
    private EditText ageText;
    private EditText heightText;
    public static final String PREFERENCE_NAME= "SaveSetting";
    public static int MODE= Context.MODE_WORLD_READABLE+ Context.MODE_WORLD_WRITEABLE;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        nameText= (EditText)findViewById(R.id.name);
        ageText= (EditText)findViewById(R.id.age);
        heightText= (EditText)findViewById(R.id.height);
    }

    @Override
    public void onStart(){
        super.onStart();
        loadSharedPreferences();
    }
    @Override
    public void onStop(){
        super.onStop();
        saveSharedPreferences();
    }

    private void loadSharedPreferences(){
      SharedPreferences sharedPreferences= getSharedPreferences(PREFERENCE_ NAME, MODE);
```

```
            String name= sharedPreferences.getString("Name","Tom");
            int age= sharedPreferences.getInt("Age", 20);
            float height= sharedPreferences.getFloat("Height",1.81f);

            nameText.setText(name);
            ageText.setText(String.valueOf(age));
            heightText.setText(String.valueOf(height));
        }

        private void saveSharedPreferences(){
        SharedPreferences sharedPreferences= getSharedPreferences(PREFERENCE_NAME, MODE);
            SharedPreferences.Editor editor= sharedPreferences.edit();

            editor.putString("Name", nameText.getText().toString());
            editor.putInt("Age", Integer.parseInt(ageText.getText().toString()));
            editor.putFloat("Height", Float.parseFloat(heightText.getText().toString()));
            editor.commit();
        }
    }
```

示例 SharePreferenceDemo 将说明如何读取其他应用程序保存的 SharedPreferences 数据。SharePreferenceDemo 示例的用户界面如图 8.3 所示，示例将读取 SimplePreferenceDemo 示例保存的信息，并在程序启动时显示在用户界面上。




**图 8.3 SharePreferenceDemo 用户界面**

下面给出 SharePreferenceDemo 示例的核心代码：

```
public static final String PREFERENCE_PACKAGE= "edu.hrbeu.SimplePreferenceDemo";
public static final String PREFERENCE_NAME= "SaveSetting";
public static int MODE= Context.MODE_WORLD_READABLE+
Context.MODE_WORLD_WRITEABLE;

public void onCreate(Bundle savedInstanceState) {
    Context c= null;
    try {
    c= this.createPackageContext(PREFERENCE_PACKAGE, Context.CONTEXT_IGNORE_SECURITY);
    } catch (NameNotFoundException e) {
        e.printStackTrace();
    }
    SharedPreferences sharedPreferences= c.getSharedPreferences(PREFERENCE_NAME, MODE);
    String name= sharedPreferences.getString("Name","Tom");
    int age= sharedPreferences.getInt("Age", 20);
    float height= sharedPreferences.getFloat("Height",1.81f);
```

```
16    }
```

从上面的代码中可以看出，第 8 行代码调用了 createPackageContext()获取到了 SimplePreferenceDemo 示例的 Context，第 1 个参数是 SimplePreferenceDemo 的包名称，在代码第 1 行进行了定义，第 2 个参数 Context.CONTEXT_IGNORE_SECURIT 表示忽略所有可能产生的安全问题。这段代码可能引发异常，因此必须防止在 try/catch 中。在代码第 12 行，通过 Context 得到了 SimplePreferenceDemo 示例的 SharedPreferences 对象，同样在 getSharedPreferences()函数中，需要将正确的 SharedPreferences 名称传递给函数。

由此可见，访问其他应用程序的 SharedPreferences 必须满足三个条件：(1)共享者需要将 SharedPreferences 的访问模式设置为全局读或全局写；(2)访问者需要知道共享者的包名称和 SharedPreferences 的名称，以通过 Context 获得 SharedPreferences 对象；(3)访问者需要确切知道每个数据的名称和数据类型，用以正确读取数据。

## 8.2 文件存储

虽然 SharedPreferences 能够为程序开发人员简化数据存储和访问的过程，但使用文件系统直接保存数据仍然是数据存储中不可或缺的重要组成部分。Android 使用的是基于 Linux 的文件系统，程序开发人员可以建立和访问程序自身的私有文件，也可以访问保存在资源目录中的原始文件和 XML 文件，还可以在 SD 卡等外部存储设备中保存文件。

### 8.2.1 内部存储

Android 系统允许应用程序创建仅能够自身访问的私有文件，文件保存在设备的内部存储器上，在 Linux 系统下的/data/data/<package name>/files 目录中。Android 系统不仅支持标准 Java 的 IO 类和方法，还提供了能够简化读写流式文件过程的函数。下面主要介绍两个函数 openFileOutput()和 openFileInput()。

openFileOutput()函数为写入数据做准备而打开的应用程序私文件，如果指定的文件不存在，则创建一个新的文件。openFileOutput()函数的语法格式如下：

```
public FileOutputStream openFileOutput(String name, int mode)
```

第 1 个参数是文件名称，这个参数不可以包含描述路径的斜杠。第 2 个参数是操作模式，Android 系统支持 4 种文件操作模式，文件操作模式的说明参见表 8.1。函数的返回值是 FileOutputStream 类型。

**表 8.1　4 种文件操作模式**

| 模　式 | 说　明 |
| --- | --- |

| | |
|---|---|
| MODE_PRIVATE | 私有模式，缺陷模式，文件仅能够被文件创建程序访问，或具有相同 UID 的程序访问 |
| MODE_APPEND | 追加模式，如果文件已经存在，则在文件的结尾处添加新数据 |
| MODE_WORLD_READABLE | 全局读模式，允许任何程序读取私有文件 |
| MODE_WORLD_WRITEABLE | 全局写模式，允许任何程序写入私有文件 |

使用 openFileOutput()函数建立新文件的示例代码如下：

```
String FILE_NAME= "fileDemo.txt";
FileOutputStream fos= openFileOutput(FILE_NAME,Context.MODE_PRIVATE)
String text= "Some data";
fos.write(text.getBytes());
fos.flush();
fos.close();
```

代码首先定义的建立文件的名称 fileDemo.txt，然后使用 openFileOutput()函数以私有模式建立文件，并调用 write()函数将数据写入文件，调用 flush()函数将所有剩余的数据写入文件，最后调用 close()函数关闭 FileOutputStream。为了提高文件系统的性能，一般调用 write()函数时，如果写入的数据量较小，系统会把数据保存在数据缓冲区中，等数据量累积到一定程度时再一次性地写入文件中。因此，在调用 close()函数关闭文件前，务必要调用 flush()函数，将缓冲区内所有的数据写入文件。

openFileInput()函数为读取数据做准备而打开的应用程序私文件。openFileOutput()函数的语法格式如下：

```
public FileInputStream openFileInput (String name)
```

第 1 个参数也是文件名称，同样不允许包含描述路径的斜杠。使用 openFileInput()函数打开已有文件的示例代码如下：

```
String FILE_NAME= "fileDemo.txt";
FileInputStream fis= openFileInput(FILE_NAME);

byte[] readBytes= new byte[fis.available()];
while(fis.read(readBytes) != -1){
}
```

上面的两部分代码在实际使用过程中会遇到错误提示，因为文件操作可能会遇到各种问题而最终导致操作失败，因此代码应该使用 try/catch 捕获可能产生的异常。




在 InternalFileDemo 示例用来演示在内部存储器上进行文件写入和读取的示例。用户界面如图 8.4 所示，用户将需要写入的数据添加在 EditText 中，通过“写入文件”按钮将数据写入到/data/data/edu.hrbeu.InternalFileDemo/files/fileDemo.txt 文件中。

如果用户选择“追加模式”，数据将会添加到 fileDemo. txt 文件的结尾处。通过“读取文件”按钮，程序会自动读取 fileDemo. txt 文件的内容，并显示在界面下方的白色区域中。

下面给出 InternalFileDemo 示例的核心代码：

```
OnClickListenerwriteButtonListener= new OnClickListener() {
    @Override
    public void onClick(View v) {
        FileOutputStream fos= null;
        try {
            if (appendBox.isChecked()){
                fos= openFileOutput(FILE_NAME,Context.MODE_APPEND);
            }else {
                fos= openFileOutput(FILE_NAME,Context.MODE_PRIVATE);
            }
        String text= entryText.getText().toString();
        fos.write(text.getBytes());
        labelView.setText("文件写入成功,写入长度："+ text.length());
        entryText.setText("");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally{
            if (fos != null){
                try {
                    fos.flush();
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
```

```java
        }
    };
    OnClickListener readButtonListener= new OnClickListener() {
        @Override
          public void onClick(View v) {
                displayView.setText("");
                FileInputStream fis= null;
                try {
                  fis= openFileInput(FILE_NAME);
                  if (fis.available()==0){
                      return;
                  }
                  byte[] readBytes= new byte[fis.available()];
                  while(fis.read(readBytes) != -1){
                  }
                  String text= new String(readBytes);
                  displayView.setText(text);
                  labelView.setText("文件读取成功,文件长度: "+ text.length());
              } catch (FileNotFoundException e) {
                  e.printStackTrace();
              }
              catch (IOException e) {
                  e.printStackTrace();
              }
        }
    };
```

程序运行后,在/data/data/edu.hrbeu.InternalFileDemo/files/目录下,找到了新建立的 fileDemo.txt 文件,如图 8.5 所示。fileDemo.txt 从文件权限上进行分析,“-rw-rw---”表明文件仅允许文件创建者和同组用户读写,其他用户无权使用。文件的大小为 9 个字节,保存的数据为 Some data。

| | | | | |
|---|---|---|---|---|
| edu.hrbeu.InternalFileDemo | | 2009-07-16 | 02:28 | drwxr-xr-x |
| files | | 2009-07-16 | 02:31 | drwxrwx--x |
| fileDemo.txt | 9 | 2009-07-16 | 03:24 | -rw-rw---- |
| lib | | 2009-07-16 | 02:28 | drwxr-xr-x |

**图 8.5 fileDemo.txt 文件**

## 8.2.2 外部存储

Android 的外部存储设备指的是 SD 卡(Secure Digital Memory Card),是一种广泛使用于数码设备上的记忆卡,如图 8.6 所示。虽然并不是所有的 Android 手机都有 SD 卡,但 Android 系统提供了对 SD 卡的便捷的访问方法。

SD 卡适用于保存大尺寸的文件或者是一些无需设置访问权限的文件。如果用户希

望保存录制的视频文件和音频文件，因为设备的内部存储空间有限，所以 SD 卡则是非常适合的选择。另一方面，如果希望设置文件的访问权限，则不能够使用 SD 卡，因为 SD 卡使用的是 FAT(File Allocation Table)的文件系统，不支持访问模式和权限控制，但内部存储器使用的是 Linux 文件系统，可以通过文件访问权限的控制保证文件的私密性。

Android 模拟器支持 SD 卡，但模拟器中没有默认的 SD 卡，开发人员必须在模拟器中手工添加 SD 卡的映像文件。使用＜Android SDK＞/tools 目录下的 mksdcard 工具创建 SD 卡映像文件，命令如下：

```
mksdcard -l SDCARD 256M E:\android\sdcard_file
```

第 1 个参数-l 表示后面的字符串是 SD 卡的标签，这个新建立的 SD 卡的标签是 SDCARD。第 2 个参数 256M 表示 SD 卡的容量是 256MB。最后一个参数表示 SD 卡映像文件的保存位置，上面的命令将映像保存在 E:\android 目录下的 sdcard_file 文件中。在 CMD 中执行该命令后，则可在所指定的目录中找到生产的 SD 卡映像文件。

如果希望 Android 模拟器启动时能够自动加载指定的 SD 卡，还需要在模拟器的“运行设置”(Run Configurations)中添加 SD 卡加载命令。SD 卡加载命令中只要指明映像文件位置即可，命令如图 8.7 所示。




图 8.6　SD 卡

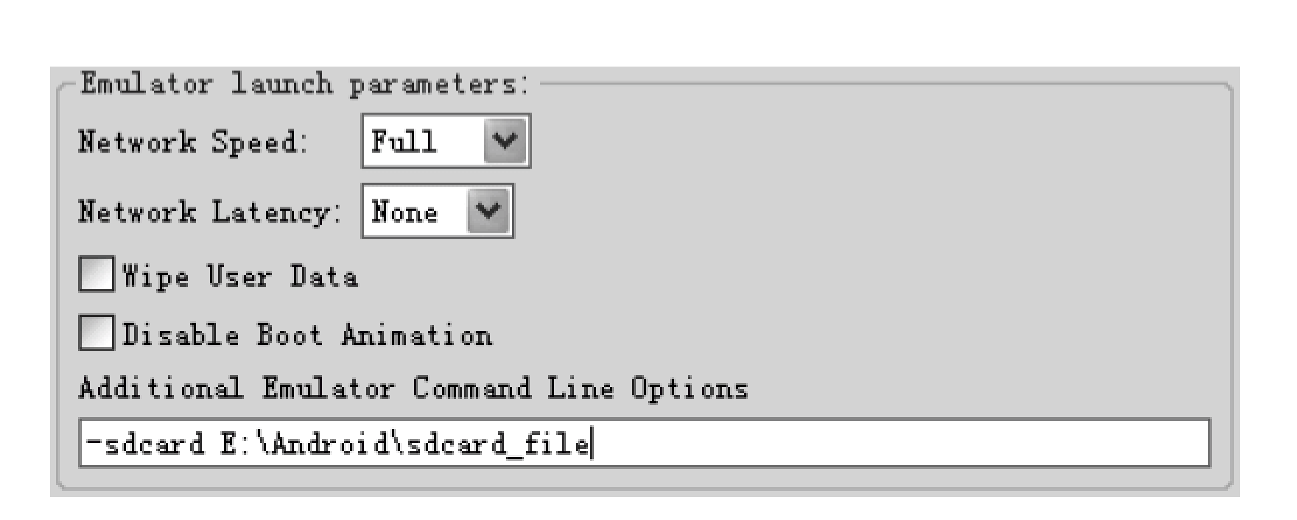



图 8.7　SD 卡加载命令

为了测试 SD 卡映像是否正确加载，在模拟器启动后，使用 FileExplorer 向 SD 卡中随意上传一个文件，如果文件上传成功，则表明 SD 卡映像已经成功加载。图 8.8 中向 SD 卡中成功上传了一个测试文件 test.txt，文件显示在/sdcard 目录下。

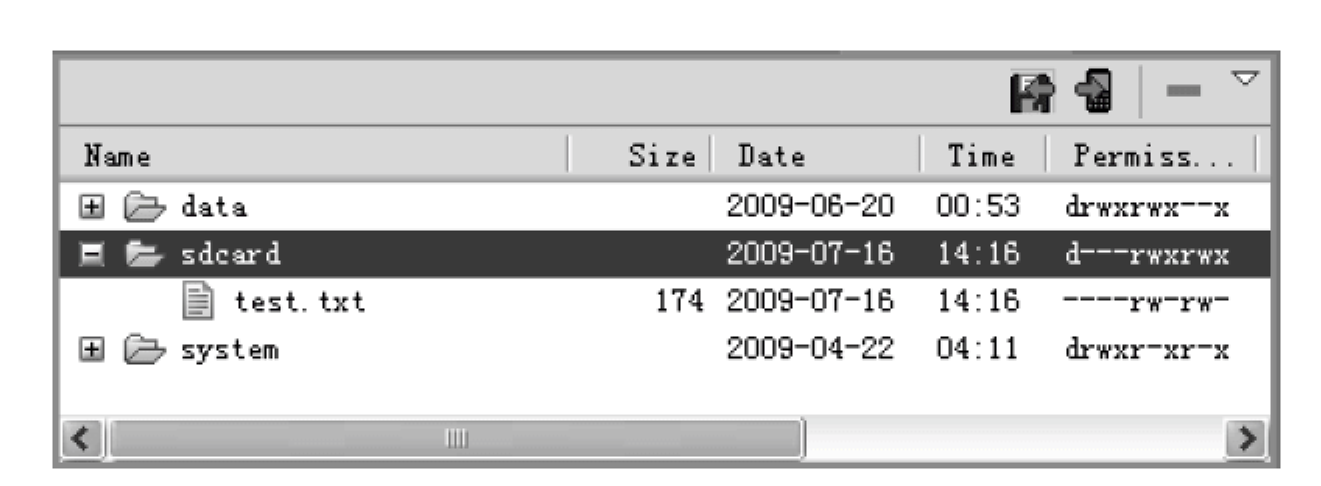



图 8.8　SD 卡加载

在 Android 系统编程访问 SD 卡非常简单。首先需要检测系统的/sdcard 目录是否可用，如果不可用，则说明设备中的 SD 卡已经被移除，在 Android 模拟器则表明 SD 卡映像没有被正确加载。如果可用，则直接通过使用标准的 Java.io.File 类进行访问。

SDcardFileDemo 示例说明了如何将数据保存在 SD 卡中。用户首先通过“生产随机

数列”按钮生产 10 个随机小数，然后通过“写入 SD 卡”按钮将生产的数据保存在 SD 卡的目录下。SDcardFileDemo 示例的用户界面如图 8.9 所示。




图 8.9 SDcardFileDemo 用户界面

SDcardFileDemo 示例运行后，在每次单击“写入 SD 卡”按钮后，都会在 SD 卡中生产一个新文件，文件名各不相同，如图 8.10 所示。

SDcardFileDemo 示例与 InternalFileDemo 示例的核心代码比较相似，不同之处在于代码中添加了 /sdcard 目录存在性检查（代码第 7 行），并使用“绝对目录＋文件名”的形式表示新建立的文件（代码第 8 行），并写入文件前对文件的存在性和可写入性进行检查（代码第 12 行）。为了保证在 SD 卡中多次写入时文件名不会重复，在文件名中使用了唯一且不重复的标识（代码第 5 行），这个标识通过调用 System.currentTimeMillis() 函数获得，表示从 1970 年 00:00:00 到当前所经过的毫秒数。SDcardFileDemo 示例的核心代码如下：

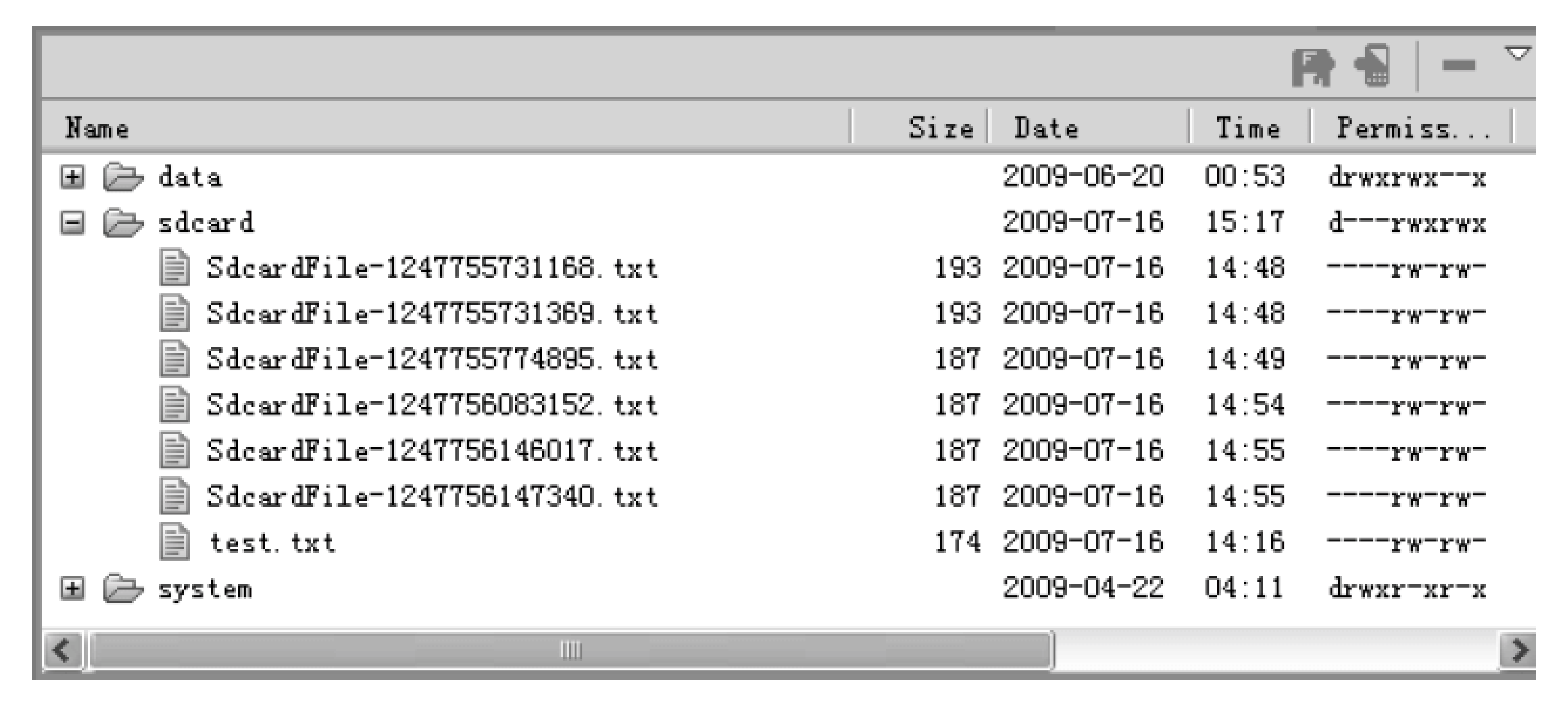



图 8.10 SD 卡中生产的文件

```
1    private static String randomNumbersString= "";
2    OnClickListener writeButtonListener= new OnClickListener() {
3        @Override
4        public void onClick(View v) {
5            String fileName= "SdcardFile- "+ System.currentTimeMillis()+ ".txt";
6            File dir= new File("/sdcard/");
7            if (dir.exists() && dir.canWrite()) {
8                File newFile= new File(dir.getAbsolutePath()+ "/"+ fileName);
9                FileOutputStream fos= null;
10               try {
11                   newFile.createNewFile();
12                   if (newFile.exists() && newFile.canWrite()) {
13                       fos= new FileOutputStream(newFile);
14                       fos.write(randomNumbersString.getBytes());
15                       TextView labelView= (TextView)findViewById(R.id.label);
```

```
                    labelView.setText(fileName+"文件写入 SD卡");
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (fos != null) {
                    try{
                        fos.flush();
                        fos.close();
                    }
                    catch (IOException e) { }
                }
            }
        }
    }
};
```

### 8.2.3 资源文件

程序开发人员除了可以在内部存储设备和外部存储设备上使用文件以外，还可以将程序开发阶段已经准备好的原始格式文件和 XML 文件分别存放在/res/raw 和/res/xml 目录下，供应用程序在运行时进行访问。

原始格式文件可以是任何格式的文件，例如视频格式文件、音频格式文件、图像文件和数据文件等等，在应用程序编译和打包时，/res/raw 目录下的所有文件都会保留原有格式不变。而/res/xml 目录下的 XML 文件，一般用来保存格式化的数据，在应用程序编译和打包时会将 XML 文件转换为高效的二进制格式，应用程序运行时会以特殊的方式进行访问。

ResourceFileDemo 示例演示了如何在程序运行时访问资源文件。当用户单击“读取原始文件”按钮时，程序将读取/res/raw/raw_file. txt 文件，并将内容显示在界面上，如图 8. 11(a)所示。当用户单击“读取 XML 文件”按钮时，程序将读取/res/xml/people. xml 文件，并将内容显示在界面上，如图 8. 11(b)所示。




(a) 读取原始文件

(b) 读取 XML 文件

**图 8. 11 ResourceFileDemo 用户界面**

读取原始格式文件，首先需要调用 getResource()函数获得资源对象，然后通过调用资源对象的 openRawResource()函数，以二进制流的形式打开指定的原始格式文件。在读取文件结束后，调用 close()函数关闭文件流。

ResourceFileDemo 示例中关于读取原始格式文件的核心代码如下：

```
1   Resources resources=this.getResources();
2   InputStream inputStream=null;
3   try {
4       inputStream=resources.openRawResource(R.raw.raw_file);
5       byte[] reader=new byte[inputStream.available()];
6       while (inputStream.read(reader) != -1) {
7       }
8       displayView.setText(new String(reader,"utf-8"));
9   } catch (IOException e) {
10      Log.e("ResourceFileDemo", e.getMessage(), e);
11  } finally {
12      if (inputStream != null) {
13      try {
14          inputStream.close();
15      }
16      catch (IOException e) { }
17      }
18  }
```

代码第 8 行的 new String(reader,"utf-8")，表示以 UTF-8 的编码方式，从字节数组中实例化一个字符串。程序开发人员需要确定/res/raw/raw_file.txt 文件使用的是 UTF-8 编码方式，否则程序运行时会产生乱码。确认的方法是在 raw_file.txt 文件上右击，选择 Properties，打开 raw_file.txt 文件的属性设置框，然后在 Resource 栏下的 Text file encoding 中，选择 Other 为 UTF-8，如图 8.12 所示。

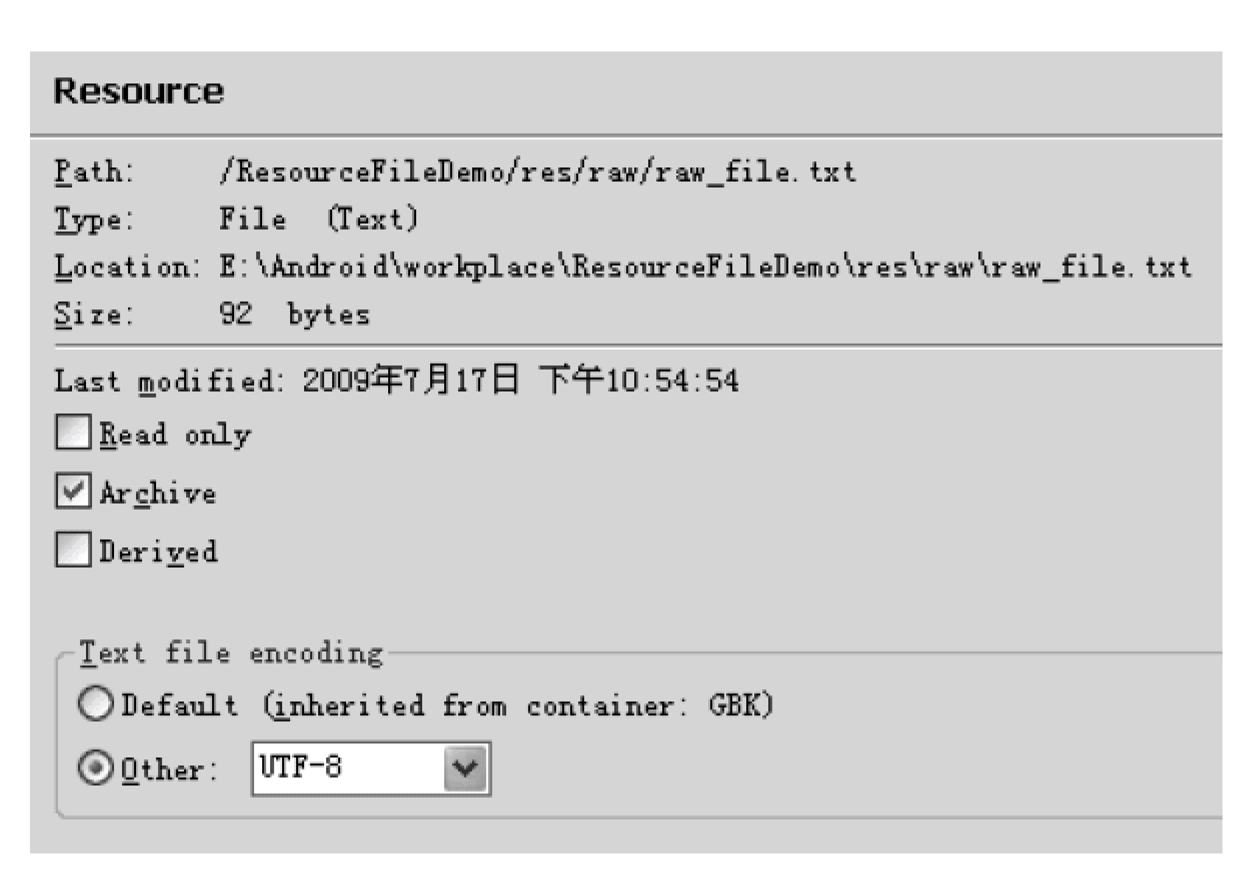



**图 8.12 更改 raw_file.txt 文件编码方式**

/res/xml 目录下的 XML 文件，与其他资源文件有所不同，程序开发人员不能够以流的方式直接读取，其主要原因在于 Android 系统为了提高读取效率，减少占用的存储空

间，将XML文件转换为一种高效的二进制格式。

为了说明如何在程序运行时读取/res/xml目录下的XML文件，首先在/res/xml目录下创建一个名为people.xml的文件。XML文件定义了多个＜person＞元素，每个＜person＞元素都包含三个属性，即name、age和height，分别表示姓名、年龄和身高。/res/xml/people.xml文件代码如下：

```
<people>
    <person name="李某某" age="21" height="1.81" />
    <person name="王某某" age="25" height="1.76" />
    <person name="张某某" age="20" height="1.69" />
</people>
```

读取XML格式文件，首先通过调用资源对象的getXml()函数，获取到XML解析器XmlPullParser。XmlPullParser是Android平台标准的XML解析器，这项技术来自一个开源的XML解析API项目XMLPULL。

ResourceFileDemo示例中关于读取XML文件的核心代码如下：

```
XmlPullParser parser=resources.getXml(R.xml.people);
String msg="";
try {
    while (parser.next() !=XmlPullParser.END_DOCUMENT) {
        String people=parser.getName();
        String name=null;
        String age=null;
        String height=null;
        if ((people !=null) && people.equals("person")) {
            int count=parser.getAttributeCount();
            for (int i=0; i<count; i++) {
                String attrName=parser.getAttributeName(i);
                String attrValue=parser.getAttributeValue(i);
                if ((attrName !=null) && attrName.equals("name")) {
                    name=attrValue;
                } else if ((attrName !=null) && attrName.equals("age")) {
                    age=attrValue;
                }else if((attrName !=null) && attrName.equals("height")){
                    height=attrValue;
                }
            }
            if ((name !=null) && (age !=null) && (height !=null)) {
            msg+="姓名："+name+",年龄："+age+",身高："+height+"\n";
            }
        }
    }
} catch (Exception e) {
```

```
28         Log.e("ResourceFileDemo", e.getMessage(), e);
29     }
30     displayView.setText(msg);
```

代码第 1 行通过资源对象的 getXml()函数获取到 XML 解析器。第 4 行的 parser.next()方法可以获取到高等级的解析事件,并通过对比确定事件类型,XML 事件类型参考表 8.2。

**表 8.2　XmlPullParser 的 XML 事件类型**

| 事 件 类 型 | 说　　明 | 事 件 类 型 | 说　　明 |
|---|---|---|---|
| START_TAG | 读取到标签开始标志 | END_TAG | 读取到标签结束标志 |
| TEXT | 读取文本内容 | END_DOCUMENT | 文档末尾 |

第 5 行使用 getName()函数获得元素的名称,第 10 行使用 getAttributeCount()函数获取元素的属性数量,第 12 行通过 getAttributeName()函数得到属性名称。最后在第 14 行到第 19 行代码中,通过分析属性名获取到正确的属性值,并在第 23 行将属性值整理成需要显示的信息。

# 8.3　数据库存储

## 8.3.1　SQLite 数据库

SQLite 是一个开源的嵌入式关系数据库,2000 年由 D. Richard Hipp 发布。自几十年前出现的商业应用程序以来,数据库就成为了应用程序的主要组成部分,同时数据库管理系统也变得非常庞大和复杂,并占用了相当多的系统资源。随着嵌入式应用程序的大量出现,一种新型的轻量级数据库 SQLite 也随之产生。SQLite 数据库比传递的数据库更加适用于嵌入式系统,因为它占用空间非常少,运行高效可靠,可移植性好,并且提供了零配置(zero-configuration)运行模式。

SQLite 数据库的优势在于其嵌入到使用它的应用程序中。这样不仅提高了运行效率,而且屏蔽了数据库使用和管理的复杂性,程序仅需要进行最基本的数据操作,其他操作可以交给进程内部的数据库引擎完成。同时因为客户端和服务器在同一进程空间运行,不需要进行网络配置和管理,因此减少了网络调用所造成的额外开销,简化的数据库的管理过程,使应用程序更加易于部署和使用。程序开发人员仅需要把 SQLite 数据库正确编译到应用程序中即可。

SQLite 数据库采用了模块化设计,模块将复杂的查询过程分解为细小的工作进行处理。SQLite 数据库由 8 个独立的模块构成,这些独立模块又构成了三个主要的子系统。SQLite 数据库体系结构如图 8.13 所示。

接口由 SQLite C API 组成,因此无论是应用程序、脚本,还是库文件,最终都是通过接口与 SQLite 交互。

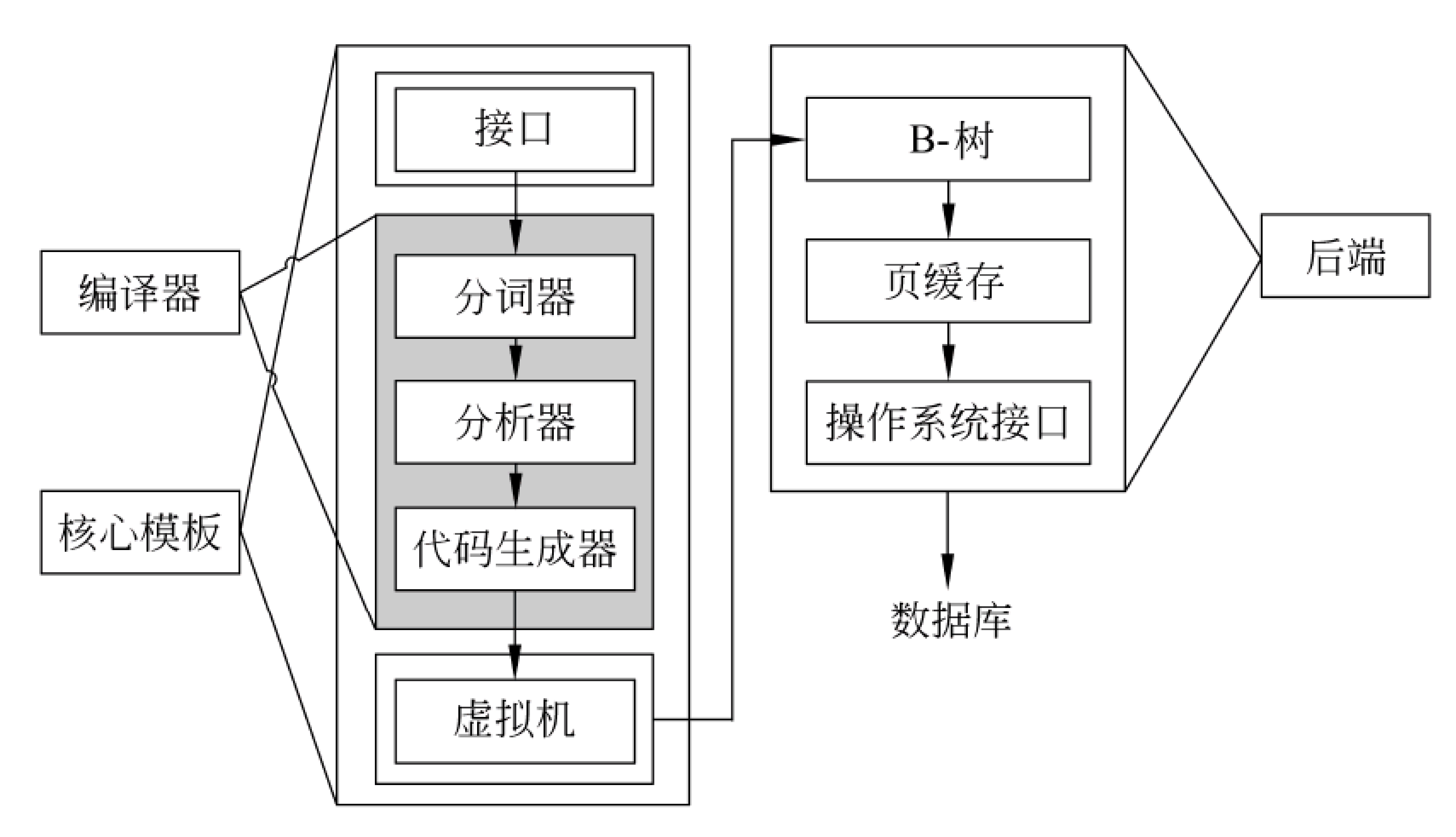



**图 8.13　SQLite 数据库体系结构**

在编译器中，分词器和分析器对 SQL 语句进行语法检查，然后把 SQL 语句转化为底层能更方便处理的分层的数据结构，这种分层的数据结构称为"语法树"。然后把语法树传给代码生成器进行处理，生成一种针对 SQLite 的汇编代码，最后由虚拟机执行。

SQLite 数据库体系结构中最核心的部分是虚拟机，也称为虚拟数据库引擎（Virtual Database Engine，VDBE）。与 Java 虚拟机相似，虚拟数据库引擎用来解释执行字节代码。虚拟数据库引擎的字节代码由 128 个操作码构成，这些操作码主要用以对数据库进行操作，每一条指令都可以完成特定的数据库操作，或以特定的方式处理栈的内容。

后端由 B-树、页缓存和操作系统接口构成。B-树和页缓存共同对数据进行管理。B-树的主要功能就是索引，它维护着各个页面之间的复杂的关系，便于快速找到所需数据。页缓存的主要作用就是通过操作系统接口在 B-树和磁盘之间传递页面。

SQLite 数据库具有很强的移植性，可以运行在 Windows，Linux，BSD，Mac OS X 和一些商用 UNIX 系统，比如 Sun 公司的 Solaris，IBM 公司的 AIX。同样，也可以工作在许多嵌入式操作系统下，例如 QNX，VxWorks，Palm OS，Symbin 和 Windows CE。SQLite 的核心大约有 3 万行标准 C 代码，模块化的设计使这些代码更加易于理解。

## 8.3.2 手动建库

在 Android 系统中，每个应用程序的 SQLite 数据库被保存在各自的/data/data/<package name>/databases 目录下。默认情况下，所有数据库都是私有的，仅允许创建数据库的应用程序访问，如果需要共享数据库则可以使用 ContentProvider。虽然应用程序完全可以在代码中动态建立 SQLite 数据库，但通过命令行手工建立和管理数据库仍然是非常重要的内容，对于调试具有数据库的应用程序非常有用。

手动建立数据库指的是使用 sqlite3 工具，通过手工输入命令行完成数据库的建立过程。sqlite3 是 SQLite 数据库自带的一个基于命令行的 SQL 命令执行工具，并可以显示命令执行结果。sqlite3 工具被集成在 Android 系统中，用户在 Linux 的命令行界面中输入 sqlite3 可启动 sqlite3 工具，并得到工具的版本信息，如下面的代码所示。启动 Linux 的命令行界面的方法是在 CMD 中输入 adb shell 命令。

```
# sqlite3
SQLite version 3.5.9
Enter ".help" for instructions
sqlite>
```

在启动 sqlite3 工具后，提示符从“#”变为“sqlite>”，表示命令行界面进入与 SQLite 数据库的交互模式，此时可以输入命令建立、删除或修改数据库的内容。正确退出 sqlite3 工具的方法是使用.exit 命令。

```
sqlite> .exit
#
```

原则上，每个应用程序的数据库都保存在各自的/data/data/<package name>/databases 目录下，但如果使用手工方式建立数据库，则必须手工建立数据库目录，目前版本无须修改数据库目录的权限。

```
# mkdir databases
# ls -l
drwxrwxrwx root      root          2009-07-18 15:43 databases
drwxr-xr-x system    system        2009-07-18 15:31 lib
#
```

在 SQLite 数据库中，每个数据库保存在一个独立的文件中，使用 sqlite3 工具后加文件名的方式打开数据库文件，如果指定的文件不存在，sqlite3 工具则自动创建新文件。下面的代码将创建名为 people 的数据库，在文件系统中将产生一个名为 people.db 的数据库文件。

```
# sqlite3 people.db
SQLite version 3.5.9
Enter ".help" for instructions
sqlite>
```

下面的代码在新创建的数据库中，构造了一个名为 peopleinfo 的表，使用 create table 命令，关系模式为 peopleinfo（_id，name，age，height）。表包含 4 个属性，_id 是整型的主键；name 表示姓名，字符型；not null 表示这个属性一定要填写，不可以为空值；age 表示年龄，整数型；height 表示身高，浮点型。

```
sqlite> create table peopleinfo
...> (_id integer primary key autoincrement,
...> name text not null,
...> age integer,
...> height float);
sqlite>
```

为了确认数据表是否创建成功，可以使用.tables 命令，显示当前数据库中的所有表。从下面的代码中可以观察到，当前数据库仅有一个名为 peopleinfo 的表。

```
sqlite> .tables
poepleinfo
sqlite>
```

当然,也可以使用.schema命令查看建立表时使用的SQL命令。如果当前数据库中包含多个表,则可以使用[.schema 表名]的形式,显示指定表的建立命令。

```
sqlite> .schema
CREATE TABLE peopleinfo (_id integer primary key autoincrement,
name text not null, age integer, height float);
sqlite>
```

下一步是向peopleinfo表中添加数据,使用insert into ... values命令。在下面的代码运行成功后,数据库的peopleinfo表将有三条数据,内容如表8.3所示。因为_id是自增加的主键,因此在输入null后,SQLite数据库会自动填写该项的内容。

```
sqlite> insert into peopleinfo values(null,'Tom',21,1.81);
sqlite> insert into peopleinfo values(null,'Jim',22,1.78);
sqlite> insert into peopleinfo values(null,'Lily',19,1.68);
```

**表8.3 peopleinfo表内容**

| _id | name | age | height |
|---|---|---|---|
| 1 | Tom | 21 | 1.81 |
| 2 | Jim | 22 | 1.78 |
| 3 | Lily | 19 | 1.68 |

在数据添加完毕后,使用select命令,显示指定数据表中的所有数据信息,命令格式为[select 属性 from 表名]。下面的代码用来显示peopleinfo表的所有数据。

```
select * from peopleinfo;
1|Tom|21|1.81
2|Jim|22|1.78
3|Lily|19|1.68
sqlite>
```

上面的查询结果看起来不是非常直观,可以使用mode命令将结果输出格式更改为"表格"方式。mode命令除了支持常见的column格式,还支持csv格式、html格式、insert格式、line格式、list格式、tabs格式和tcl格式。

```
sqlite> .mode column
sqlite> select * from peopleinfo;
1          Tom        21         1.81
2          Jim        22         1.78
3          Lily       19         1.68
sqlite>
```

更新数据可以使用 update 命令，命令格式为[update 表名 set 属性="新值" where 条件]。更新数据后，同样使用 select 命令显示数据，则可以确定数据是否正确更新。下面的代码将姓名为 Lily 数据中的高度值更新为 1.88。

```
sqlite> update peopleinfo set height=1.88 where name="Lily";
sqlite> select * from peopleinfo;
select * from peopleinfo;
1           Tom         21          1.81
2           Jim         22          1.78
3           Lily        19          1.88
sqlite>
```

删除数据可以使用 delete 命令，命令格式为[delete from 表名 where 条件]。下面的代码将_id 为 3 的数据从表 peopleinfo 中删除。

```
sqlite> delete from peopleinfo where _id=3;
sqlite> select * from peopleinfo;
select * from peopleinfo;
1           Tom         21          1.81
2           Jim         22          1.78
sqlite>
```

sqlite3 工具还支持大量的命令，可以使用.help 命令查询 sqlite3 的命令列表，也可以参考表 8.4。

**表 8.4 sqlite3 命令列表**

| 编号 | 命令 | 说明 |
|---|---|---|
| 1 | .bail ON\|OFF | 遇到错误时停止，默认为 OFF |
| 2 | .databases | 显示数据库名称和文件位置 |
| 3 | .dump ? TABLE? ... | 将数据库以 SQL 文本形式导出 |
| 4 | .echo ON\|OFF | 开启和关闭回显 |
| 5 | .exit | 退出 |
| 6 | .explain ON\|OFF | 开启或关闭适当输出模式，如果开启模式将更改为 column，并自动设置宽度 |
| 7 | .header(s) ON\|OFF | 开启或关闭标题显示 |
| 8 | .help | 显示帮助信息 |
| 9 | .import FILE TABLE | 将数据从文件导入表中 |
| 10 | .indices TABLE | 显示表中所的列名 |
| 11 | .load FILE ? ENTRY? | 导入扩展库 |
| 12 | .mode MODE ? TABLE? | 设置输入格式 |
| 13 | .nullvalue STRING | 打印时使用 STRING 代替 NULL |

续表

| 编　号 | 命　令 | 说　明 |
| --- | --- | --- |
| 14 | .output FILENAME | 将输入保存到文件 |
| 15 | .output stdout | 将输入显示在屏幕上 |
| 16 | .prompt MAIN CONTINUE | 替换标准提示符 |
| 17 | .quit | 退出 |
| 18 | .read FILENAME | 在文件中执行 SQL 语句 |
| 19 | .schema ? TABLE? | 显示表的创建语句 |
| 20 | .separator STRING | 更改输入和导入的分隔符 |
| 21 | .show | 显示当前设置变量值 |
| 22 | .tables ? PATTERN? | 显示符合匹配模式的表名 |
| 23 | .timeout MS | 尝试打开被锁定的表 MS 毫秒 |
| 24 | .timer ON\|OFF | 开启或关闭 CPU 计时器 |
| 25 | .width NUM NUM… | 设置 column 模式的宽度 |

### 8.3.3 代码建库

在代码中动态建立数据库是比较常用的方法。例如在程序运行过程中，当需要进行数据库操作时，应用程序会首先尝试打开数据库，此时如果数据库并不存在，程序则会自动建立数据库，然后再打开数据库。

在编程实现时，一般将所有对数据库的操作都封装在一个类中，因此只要调用这个类，就可以完成对数据库的添加、更新、删除和查询等操作。下面内容是 DBAdapter 类的部分代码，封装了数据库的建立、打开和关闭等操作：

```
public class DBAdapter {
    private static final String DB_NAME= "people.db";
    private static final String DB_TABLE= "peopleinfo";
    private static final int DB_VERSION= 1;

    public static final String KEY_ID= "_id";
    public static final String KEY_NAME= "name";
    public static final String KEY_AGE= "age";
    public static final String KEY_HEIGHT= "height";

    private SQLiteDatabase db;
    private final Context context;
    private DBOpenHelper dbOpenHelper;

    private static class DBOpenHelper extends SQLiteOpenHelper {}
```

```
16
17      public DBAdapter(Context _context) {
18          context= _context;
19      }
20
21      public void open() throws SQLiteException {
22          dbOpenHelper= new DBOpenHelper(context, DB_NAME, null, DB_VERSION);
23          try {
24              db= dbOpenHelper.getWritableDatabase();
25          }catch (SQLiteException ex) {
26              db= dbOpenHelper.getReadableDatabase();
27          }
28      }
29
30      public void close() {
31          if (db != null){
32              db.close();
33              db= null;
34          }
35      }
36  }
```

从代码的第 2 行到第 9 行可以看出，在 DBAdapter 类中首先声明了数据库的基本信息，包括数据库文件的名称、数据库表格名称和数据库版本，以及数据库表中的属性名称。从这些基本信息上不难发现，这个数据库与 8.3.2 节手动建立的数据库是完全相同的。

代码第 11 行声明了 SQLiteDatabase 对象 db。SQLiteDatabase 类封装了非常多的方法，用以建立、删除数据库，执行 SQL 命令，对数据进行管理等工作。

代码第 13 行声明了一个非常重要的帮助类 SQLiteOpenHelper，这个帮助类可以辅助建立、更新和打开数据库。虽然在代码第 21 行定义了 open()函数用来打开数据库，但 open()函数中并没有任何对数据库进行实际操作的代码，而是调用了 SQLiteOpenHelper 类的 getWritableDatabase()函数和 getReadableDatabase()函数。这个两个函数会根据数据库是否存在、版本号和是否可写等情况，决定在返回数据库对象前，是否需要建立数据库。

在代码第 30 行的 close()函数中，调用了 SQLiteDatabase 对象的 close()方法关闭数据库。这是上面的代码中，唯一的一个地方直接调用了 SQLiteDatabase 对象的方法。SQLiteDatabase 中也封装了打开数据库的函数 openDatabases()和创建数据库函数 openOrCreateDatabases()，因为代码中使用了帮助类 SQLiteOpenHelper，从而避免直接调用 SQLiteDatabase 中的打开和创建数据库的方法，简化了数据库打开过程中繁琐的逻辑判断过程。

代码第 15 行实现了内部静态类 DBOpenHelper，继承了帮助类 SQLiteOpenHelper，重载了 onCreate()函数和 onUpgrade()函数，代码如下：

```
1  private static class DBOpenHelper extends SQLiteOpenHelper {
2      public DBOpenHelper(Context context, String name, CursorFactory factory, int version){
3          super(context, name, factory, version);
4          }
5      private static final String DB_CREATE= "create table "+
6          DB_TABLE+ " ("+ KEY_ID+ " integer primary key autoincrement, "+
7          KEY_NAME+ " text not null, "+ KEY_AGE+ " integer,"+ KEY_HEIGHT+ " float);";
8
9      @Override
10     public void onCreate(SQLiteDatabase _db) {
11         _db.execSQL(DB_CREATE);
12         }
13
14     @Override
15     public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
16         _db.execSQL("DROP TABLE IF EXISTS "+ DB_TABLE);
17         onCreate(_db);
18     }
19 }
```

代码的第5行到第7行是创建表的SQL命令。代码第10行和第15行分别重载了onCreate()函数和onUpgrade()函数，这是继承SQLiteOpenHelper类必须重载的两个函数。onCreate()函数在数据库第一次建立时被调用，一般用来创建数据库中的表，并做适当的初始化工作。在代码第11行中，通过调用SQLiteDatabase对象的execSQL()方法，执行创建表的SQL命令。onUpgrade()函数在数据库需要升级时被调用，一般用来删除旧的数据库表，并将数据转移到新版本的数据库表中。在代码第16行和第17行中，为了简单起见，并没有做任何数据转移，而仅仅删除原有的表后建立新的数据库表。

程序开发人员不应直接调用onCreate()和onUpgrade()函数，而应该由SQLiteOpenHelper类来决定何时调用这两个函数。SQLiteOpenHelper类的getWritableDatabase()函数和getReadableDatabase()函数是可以直接调用的函数。getWritableDatabase()函数用来建立或打开可读写的数据库对象，一旦函数调用成功，数据库对象将被缓存，任何需要使用数据库对象时，都可以调用这个方法获取到数据库对象，但一定要在不使用时调用close()函数关闭数据库。如果保存数据库文件的磁盘空间已满，调用getWritableDatabase()函数则无法获得可读写的数据库对象，这时可以调用getReadableDatabase()函数，获得一个只读的数据库对象。

当然，如果程序开发人员不希望使用SQLiteOpenHelper类，同样可以直接创建数据库。首先调用openOrCreateDatabases()函数创建数据库对象，然后执行SQL命令建立数据库中的表和直接的关系，示例代码如下：

```
private static final String DB_CREATE= "create table "+
    DB_TABLE+ " ("+ KEY_ID+ " integer primary key autoincrement, "+
    KEY_NAME+ " text not null, "+ KEY_AGE+ " integer,"+ KEY_HEIGHT+ " float);";
```

```
public void create() {
    db.openOrCreateDatabases(DB_NAME, context.MODE_PRIVATE, null)
    db.execSQL(DB_CREATE);
}
```

## 8.3.4 数据操作

数据操作指的是对数据的添加、删除、查找和更新的操作，虽然程序开发人员完全可以通过执行 SQL 命名完成数据操作，但还是推荐使用 Android 提供的专用类和方法，这些类和方法更加简洁、易用。

为了使 DBAdapter 类支持对数据的添加、删除、更新和查找等功能，在 DBAdapter 类中增加下面的这些函数。其中，insert(People people)用来添加一条数据，queryAllData()用来获取全部数据，queryOneData(long id)根据 id 获取一条数据，deleteAllData()用来删除全部数据，deleteOneData(long id)根据 id 删除一条数据，updateOneData(long id , People people)根据 id 更新一条数据。

```
public class DBAdapter {
    public long insert(People people) {}
    public long deleteAllData() { }
    public long deleteOneData(long id) { }
    public People[] queryAllData() {}
    public People[] queryOneData(long id) { }
    public long updateOneData(long id , People people){ }

    private People[] ConvertToPeople(Cursor cursor){}
}
```

ConvertToPeople(Cursor cursor)是私有函数，作用是将查询结果转换为用来存储数据自定义的 People 类对象。People 类的包含 4 个公共属性，分别为 ID、Name、Age 和 Height，对应数据库中的 4 个属性值。重载 toString()函数，主要是便于界面显示的需要。People 类的代码如下：

```
    public class People {
        public int ID=-1;
        public String Name;
        public int Age;
        public float Height;

        @Override
        public String toString(){
            String result="";
            result+="ID: "+this.ID+",";
            result+="姓名: "+this.Name+",";
```

```
                result+="年龄："+this.Age+"，";
                result+="身高："+this.Height+",";
                return result;
        }
}
```

SQLiteDatabase 类的公共函数 insert()、delete()、update()和 query()，封装了执行的添加、删除、更新和查询功能的 SQL 命令。下面分别介绍如何使用 SQLiteDatabase 类的公共函数，完成数据的添加、删除、更新和查询等操作。

### 1. 添加功能

为了添加一条新数据，首先构造一个 ContentValues 对象，然后调用 ContentValues 对象的 put()方法，将每个属性的值写入到 ContentValues 对象中，最后使用 SQLiteDatabase 对象的 insert()函数，将 ContentValues 对象中的数据写入指定的数据库表中。insert()函数的返回值是新数据插入的位置，即 ID 值。ContentValues 类是一个数据承载容器，主要用来向数据库表中添加一条数据。

```
1   public long insert(People people) {
2       ContentValues newValues=new ContentValues();
3
4       newValues.put(KEY_NAME, people.Name);
5       newValues.put(KEY_AGE, people.Age);
6       newValues.put(KEY_HEIGHT, people.Height);
7
8       return db.insert(DB_TABLE, null, newValues);
9   }
```

第 4 行代码向 ContentValues 对象 newValues 中添加一个名称/值对，put()函数的第 1 个参数是名称，第 2 个参数是值。在第 8 行代码的 insert()函数中，第 1 个参数是数据表的名称，第 2 个参数是在 NULL 时的替换数据，第 3 个参数是需要向数据库表中添加的数据。

### 2. 删除功能

删除数据比较简单，只需要调用当前数据库对象的 delete()函数，并指明表名称和删除条件即可。

```
public long deleteAllData() {
    return db.delete(DB_TABLE, null, null);
}

public long deleteOneData(long id) {
    return db.delete(DB_TABLE,  KEY_ID+"="+id, null);
}
```

delete()函数的第1个参数是数据库的表名称,第2个参数是删除条件。在第2行代码中,删除条件为null,表示删除表中的所有数据。而代码第6行则指明了需要删除数据的id值,因此deleteOneData()函数仅删除一条数据,此时delete()函数的返回值表示被删除的数据的数量。

### 3. 更新功能

更新数据同样要使用ContentValues对象,首先构造ContentValues对象,然后调用put()函数将属性的值写入到ContentValues对象中,最后使用SQLiteDatabase对象的update()函数,并指定数据的更新条件。

```
1   public long updateOneData(long id , People people){
2       ContentValues updateValues= new ContentValues();
3       updateValues.put(KEY_NAME, people.Name);
4       updateValues.put(KEY_AGE, people.Age);
5       updateValues.put(KEY_HEIGHT, people.Height);
6
7       return db.update(DB_TABLE, updateValues,  KEY_ID+ "="+ id, null);
8   }
```

在代码的第7行中,update()函数的第1个参数表示数据表的名称,第2个参数是更新条件。update()函数的返回值表示数据库表中被更新的数据数量。

### 4. 查询功能

介绍查询功能前,先要介绍Cursor类。在Android系统中,数据库查询结果的返回值并不是数据集合的完整拷贝,而是返回数据集的指针,这个指针就是Cursor类。Cursor类支持在查询的数据集合中多种移动方式,并能够获取数据集合的属性名称和序号,具体的方法和说明可以参考表8.5。

**表8.5 Cursor类的方法和说明**

| 函　　数 | 说　　明 |
|---|---|
| moveToFirst | 将指针移动到第一条数据上 |
| moveToNext | 将指针移动到下一条数据上 |
| moveToPrevious | 将指针移动到上一条数据上 |
| getCount | 获取集合的数据数量 |
| getColumnIndexOrThrow | 返回指定属性名称的序号,如果属性不存在则产生异常 |
| getColumnName | 返回指定序号的属性名称 |
| getColumnNames | 返回属性名称的字符串数组 |

续表

| 函　　数 | 说　　明 |
|---|---|
| getColumnIndex | 根据属性名称返回序号 |
| moveToPosition | 将指针移动到指定的数据上 |
| getPosition | 返回当前指针的位置 |

从 Cursor 中提取数据可以参考 ConvertToPeople() 函数的实现方法。在提取 Cursor 数据中的数据前，推荐测试 Cursor 中的数据数量，避免在数据获取中产生异常，例如代码的第 3 行到第 5 行。从 Cursor 中提取数据使用安全类型的 get<Type>() 函数，函数的输入值为属性的序号，为了获取属性的序号，可以使用 getColumnIndex() 函数获取指定属性的序号。

```
1    private People[] ConvertToPeople(Cursor cursor){
2        int resultCounts= cursor.getCount();
3        if (resultCounts==0 || !cursor.moveToFirst()){
4            return null;
5        }
6        People[] peoples= new People[resultCounts];
7        for (int i=0; i< resultCounts; i++){
8            peoples[i]= new People();
9            peoples[i].ID= cursor.getInt(0);
10           peoples[i].Name= cursor.getString(cursor.getColumnIndex(KEY_NAME));
11           peoples[i].Age= cursor.getInt(cursor.getColumnIndex(KEY_AGE));
12           peoples[i].Height= cursor.getFloat(cursor.getColumnIndex(KEY_HEIGHT));
13           cursor.moveToNext();
14       }
15       return peoples;
16   }
```

要进行数据查询就需要调用 SQLiteDatabase 类的 query() 函数，这个函数的参数较多，可以参考表 8.6 参数说明，query() 函数的语法如下：

```
Cursor android.database.sqlite.SQLiteDatabase.query(String table, String[] columns, String selection,
String[] selectionArgs, String groupBy, String having, String orderBy)
```

**表 8.6　query() 函数的参数说明**

| 位置 | 类型+名称 | 说　　明 |
|---|---|---|
| 1 | String table | 表名称 |
| 2 | String[] columns | 返回的属性列名称 |

续表

| 位置 | 类型+名称 | 说　　明 |
| --- | --- | --- |
| 3 | String selection | 查询条件 |
| 4 | String[] selectionArgs | 如果在查询条件中使用问号,则需要定义替换符的具体内容 |
| 5 | String groupBy | 分组方式 |
| 6 | String having | 定义组的过滤器 |
| 7 | String orderBy | 排序方式 |

下面给出根据 id 查询数据的代码:

```
public People[] getOneData(long id) {
    Cursor results= db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT},
        KEY_ID+ "= "+ id, null, null, null, null);
    return ConvertToPeople(results);
}
```

下面是查询全部数据的代码:

```
public People[] getAllData() {
    Cursor results= db.query(DB_TABLE, new String[] { KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT},
        null, null, null, null, null);
    return ConvertToPeople(results);
}
```

SQLiteDemo 是对数据库操作的一个示例,如图 8.14 所示。在这个示例中,用户可以在界面的上方输入数据信息,通过“添加数据”按钮将数据写入数据库。“全部显示”相当于查询数据库中的所有数据,并将数据显示在界面下方。“清除显示”仅是清除界面下面显示的数据,而不对数据库进行任何操作。“全部删除”是数据库操作,将删除数据库中的所有数据。在界面中部,以“ID+功能”命名的按钮,分别是根据 ID 删除数据,根据 ID 进行数据查询,根据 ID 更新数据,而这个 ID 值就取自本行的 EditText 控件。下面不再给出 SQLiteDemo 示例的代码。




图 8.14　SQLiteDemo 用户界面

## 8.4 数据共享

### 8.4.1 ContentProvider

ContentProvider(数据提供者)是在应用程序间共享数据的一种接口机制。应用程

序在不同的进程中运行，因此，数据和文件在不同应用程序之间是不能够直接访问的。SharedPreferences 和文件存储为跨越程序边界的访问提供了方法，但这些方法都有局限性。ContentProvider 提供了更为高级的数据共享方法，应用程序可以指定需要共享的数据，而其他应用程序则可以在不知数据来源、路径的情况下，对共享数据进行查询、添加、删除和更新等操作。

Android 系统中，除了程序开发人员通过 ContentProvider 提供的共享数据外，还有许多 Android 系统的内置数据也通过 ContentProvider 提供给用户使用，例如通讯录、音视频文件和图像文件等等。

在创建 ContentProvider 时，需要首先使用数据库、文件系统或网络实现底层存储功能，然后在继承 ContentProvider 的类中实现基本数据操作的接口函数，包括添加、删除、查找和更新等功能。调用者不能够直接调用 ContentProvider 的接口函数，而需要使用 ContentResolver 对象，通过 URI 间接调用 ContentProvider，调用关系如图 8.15 所示。

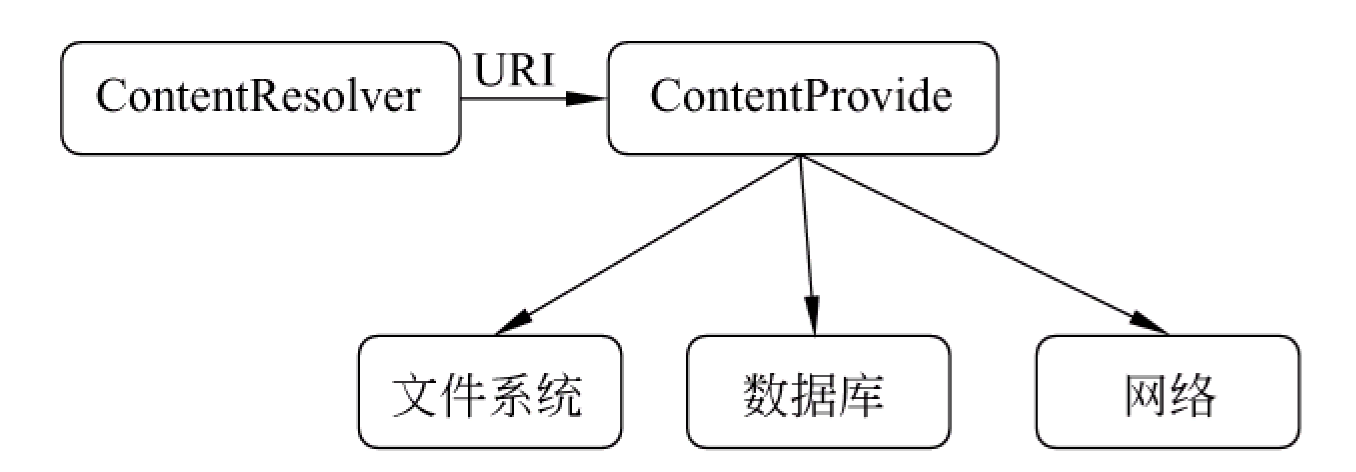



**图 8.15 ContentProvider 调用关系**

程序开发人员使用 ContentResolver 对象与 ContentProvider 进行交互，而 ContentResolver 则通过 URI 确定需要访问的 ContentProvider 的数据集。在发起一个请求的过程中，Android 首先根据 URI 确定处理这个查询的 ContentResolver，然后初始化 ContentResolver 所有需要的资源，这个初始化的工作是 Android 系统完成的，无需程序开发人员参与。一般情况下只有一个 ContentResolver 对象，但却可以同时与多个 ContentResolver 进行交互。

ContentProvider 完全屏蔽了数据提供组件的数据存储方法。在使用者看来，数据提供者通过 ContentProvider 提供了一组标准的数据操作接口，却无法得知数据提供者的数据存储方。数据提供者可以使用 SQLite 数据库存储数据，也可以通过文件系统或 SharedPreferences 存储数据，甚至是使用网络存储的方法，这些内容对数据使用者都是不可见的。同时也正是因为屏蔽数据的存储方法，很大程度上简化了 ContentProvider 的使用难度，使用者只要调用 ContentProvider 提供的接口函数，就可完成所有的数据操作，而具体的数据存储方法则是 ContentProvider 的设计者应去思考的问题。

ContentProvider 的数据模式似于数据库的数据表，每行是一条记录，每列具有相同的数据类型，如表 8.7 所示。每条记录都包含一个长型的字段_ID，用来唯一标识每条记

**表 8.7 ContentProvider 数据模型**

| _ID | NAME | AGE | HEIGHT |
|---|---|---|---|
| 1 | Tom | 21 | 1.81 |
| 2 | Jim | 22 | 1.78 |

录。ContentProvider 可以提供多个数据集，调用者使用 URI 对不同的数据集的数据进行操作。

URI 是通用资源标志符（Uniform Resource Identifier），用来定位任何远程或本地的可用资源。ContentProvider 使用的 URI 语法结构如下：

```
content://<authority>/<data_path>/<id>
```

其中，content://是通用前缀，表示该 URI 用于 ContentProvider 定位资源，无需修改。<authority>是授权者名称，用来确定具体由哪一个 ContentProvider 提供资源。因此，一般<authority>都由类的小写全称组成，以保证唯一性。<data_path>是数据路径，用来确定请求的是哪个数据集。如果 ContentProvider 仅提供一个数据集，数据路径则是可以省略的。但如果 ContentProvider 仅提供多个数据集，数据路径则必须指明具体是哪一个数据集。数据集的数据路径可以写成多段格式，例如/people/girl 和/people/boy。<id>是数据编号，用来唯一确定数据集中的一条记录，用来匹配数据集中_ID 字段的值。如果请求的数据并不只限于一条数据，则<id>是可以省略的。

例如，请求整个 people 数据集的 URI 应写为：

```
content://edu.hrbeu.peopleprovider/people
```

而请求 people 数据集中第 3 条数据的 URI 则应写为：

```
content://edu.hrbeu.peopleprovider/people/3
```

## 8.4.2 创建数据提供者

程序开发人员通过继承 ContentProvider 类可以创建一个新的数据提供者，过程可以分为三步：

(1) 继承 ContentProvider，并重载 6 个函数。

(2) 声明 CONTENT_URI，实现 UriMatcher。

(3) 注册 ContentProvider。

下面按照上述的三个步骤，逐步说明创建数据提供者的过程。

### 1. 继承 ContentProvider，并重载 6 个函数

新建立的类继承 ContentProvider 后，共有 6 个函数需要重载，分别是 delete()、getType()、insert()、onCreate()、qurey()和 update()。其中，delete()、insert()、qurey()和 update()分别用于对数据集的删除、添加、查询和更新操作，程序开发人员根据底层数据的存储方式不同，使用不同方式实现数据操作函数。而 onCreate()一般用来初始化底层数据集和建立数据连接等工作。getType()函数用来返回指定 URI 的 MIME 数据类型，如果 URI 是单条数据，则返回的 MIME 数据类型应以 vnd. android. cursor. item 开头；如果 URI 是多条数据，则返回的 MIME 数据类型应以 vnd. android. cursor. dir/开头。

新建立的类继承 ContentProvider 后，Eclipse 会提示程序开发人员需要重载部分代

码，并自动生成需要重载的代码框架。下面的代码是 Eclipse 自动生成的代码框架：

```
import android.content.* ;
import android.database.Cursor;
import android.net.Uri;

public class PeopleProvider extends ContentProvider{

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean onCreate() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;
```

```
        }
}
```

### 2. 声明 CONTENT_URI，实现 UriMatcher

在新构造的 ContentProvider 类中，经常需要判断 URI 是单条数据还是多条数据，最简单的方法是构造一个 UriMatcher。同时，为了便于判断和使用 URI，一般将 URI 的授权者名称和数据路径等内容声明为静态常量，并声明 CONTENT_URI。

声明 CONTENT_URI 和构造 UriMatcher 的代码如下：

```
1   public static final String AUTHORITY= "edu.hrbeu.peopleprovider";
2   public static final String PATH_SINGLE= "people/# ";
3   public static final String PATH_MULTIPLE= "people";
4   public static final String CONTENT_URI_STRING= "content://"+ AUTHORITY+ "/"+ PATH_MULTIPLE;
5   public static final Uri  CONTENT_URI= Uri.parse(CONTENT_URI_STRING);
6   private static final int MULTIPLE_PEOPLE= 1;
7   private static final int SINGLE_PEOPLE= 2;
8
9   private static final UriMatcher uriMatcher;
10  static {
11      uriMatcher= new UriMatcher(UriMatcher.NO_MATCH);
12      uriMatcher.addURI(AUTHORITY, PATH_SINGLE, MULTIPLE_PEOPLE);
13      uriMatcher.addURI(AUTHORITY, PATH_MULTIPLE, SINGLE_PEOPLE);
14  }
```

代码第 1 行声明了 URI 的授权者名称，第 2 行声明了单条数据的数据路径，第 3 行声明了多条数据的数据路径，第 4 行声明了 CONTENT_URI 的字符串形式，第 5 行则正式声明了 CONTENT_URI，第 6 行声明了多条数据的返回代码，第 7 行声明了单条数据的返回代码。第 9 行声明了 UriMatcher，并在第 10 行到第 13 行的静态构造函数中，声明了 UriMatcher 的匹配方式和返回代码。其中，第 11 行 UriMatcher 的构造函数中，UriMatcher.NO_MATCH 表示 URI 无匹配时的返回代码。第 12 行的 addURI()函数用来添加新的匹配项，语法如下：

```
public void  addURI  (String authority, String path, int code)
```

其中，authority 表示匹配的授权者名称，path 表示数据路径，# 可以代表任何数字，code 表示返回代码。

使用 UriMatcher 时，则可以直接调用 match()函数，对指定的 URI 进行判断，示例代码如下：

```
switch(uriMatcher.match(uri)){
    case MULTIPLE_PEOPLE:
        //多条数据的处理过程
        break;
```

```
        case SINGLE_PEOPLE:
            //单条数据的处理过程
            break;
        default:
            throw new IllegalArgumentException("不支持的 URI:"+uri);
    }
```

### 3. 注册 ContentProvider

在完成 ContentProvider 类的代码实现后，需要在 AndroidManifest.xml 文件中进行注册。注册 ContentProvider 使用<provider>标签，示例代码如下：

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <provider android:name=".PeopleProvider"
            android:authorities="edu.hrbeu.peopleprovider"/>
</application>
```

在上面的代码中，注册了一个授权者名称为 edu.hrbeu.peopleprovider 的 ContentProvider，其实现类是 PeopleProvider。

## 8.4.3 使用数据提供者

使用 ContentProvider 并不需要直接调用类中的数据操作函数，而是通过 Android 组件都具有的 ContentResolver 对象，通过 URI 进行数据操作。程序开发人员只需要知道 URI 和数据集的数据格式，则可以进行数据操作，解决不同应用程序之间的数据共享问题。

每个 Android 组件都具有一个 ContentResolver 对象，获取 ContentResolver 对象的方法是调用 getContentResolver()函数。

```
ContentResolver resolver=getContentResolver();
```

### 1. 查询操作

在获取到 ContentResolver 对象后，程序开发人员则可以使用 query()函数查询目标数据。下面的代码是查询 ID 为 2 的数据：

```
String KEY_ID="_id";
String KEY_NAME="name";
String KEY_AGE="age";
String KEY_HEIGHT="height";

Uri uri=Uri.parse(CONTENT_URI_STRING+"/"+"2";
Cursor cursor=resolver.query(uri,
    new String[] {KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT}, null, null, null);
```

从上面的代码不难看出，在 URI 中定义了需要查询数据的 ID，在 query()函数并没有额外声明查询条件。如果需要获取数据集中的全部数据，则可直接使用 CONTENT_URI，此时 ContentProvider 在分析 URI 时将认为需要返回全部数据。

ContentResolver 的 query()函数与 SQLite 数据库的 query()函数非常相似，语法结构如下：

```
Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

uri 定义了查询的数据集，projection 定义了从数据集返回哪些数据项，selection 定义了返回数据的查询条件。

### 2. 添加操作

向 ContentProvider 中添加数据有两种方法，一种是使用 insert()函数，向 ContentProvider 中添加一条数据；另一种是使用 bultInsert()函数，批量添加数据。下面的代码说明了如何使用 insert()函数添加单条数据：

```
ContentValues values=new ContentValues();
values.put(KEY_NAME, "Tom");
values.put(KEY_AGE, 21);
values.put(KEY_HEIGHT, 1.81f);

Uri newUri=resolver.insert(CONTENT_URI, values);
```

下面的代码说明了如何使用 bultInsert()函数添加多条数据：

```
ContentValues[] arrayValues=new ContentValues[10];
//实例化每一个 ContentValues
int count=resolver.bultInsert(CONTENT_URI, arrayValues);
```

### 3. 删除操作

删除操作需要使用 delete()函数。如果需要删除单条数据，则可以在 URI 中指定需要删除数据的 ID。如果需要删除多条数据，则可以在 selection 中声明删除条件。下面代码说明了如何删除 ID 为 2 的数据：

```
Uri uri=Uri.parse(CONTENT_URI_STRING+"/"+"2");
int result=resolver.delete(uri, null, null);
```

也可以在 selection 将删除条件定义为 ID 大于 4 的数据：

```
String selection=KEY_ID+">4";
int result=resolver.delete(CONTENT_URI, selection, null);
```

### 4. 更新操作

更新操作需要使用 update()函数，参数定义与 delete()函数相同，同样可以在 URI

中指定需要更新数据的 ID，也可以在 selection 中声明更新条件。下面代码说明了如何更新 ID 为 7 的数据：

```
ContentValues values= new ContentValues();
values.put(KEY_NAME, "Tom");
values.put(KEY_AGE, 21);
values.put(KEY_HEIGHT, 1.81f);

Uri uri= Uri.parse(CONTENT_URI_STRING+ "/"+ "7");
int result= resolver.update(uri, values, null, null);
```

## 8.4.4 示例

ContentProviderDemo 是一个无界面的示例，仅提供一个 ContentProvider 组件，供其他应用程序进行数据交换。底层使用 SQLite 数据库，支持数据的添加、删除、更新和查询等基本操作。ContentResolverDemo 是使用 ContentProvider 的示例，自身不具有任何数据存储功能，仅是通过 URI 访问 ContentProviderDemo 示例提供的 ContentProvider。界面如图 8.16 所示，该界面与 SQLiteDemo 示例的界面基本相同。




图 8.16 ContentResolverDemo 用户界面

从图 8.17 的文件结构上可以发现，两个示例都包含一个相同的文件 People.java，两个示例中的这个文件的内容也完全相同，定义了数据提供者和数据调用者都必须知道的信息。这些信息包括授权者名称、数据路径、MIME 数据类型、CONTENT_URI 和数据项名称等。

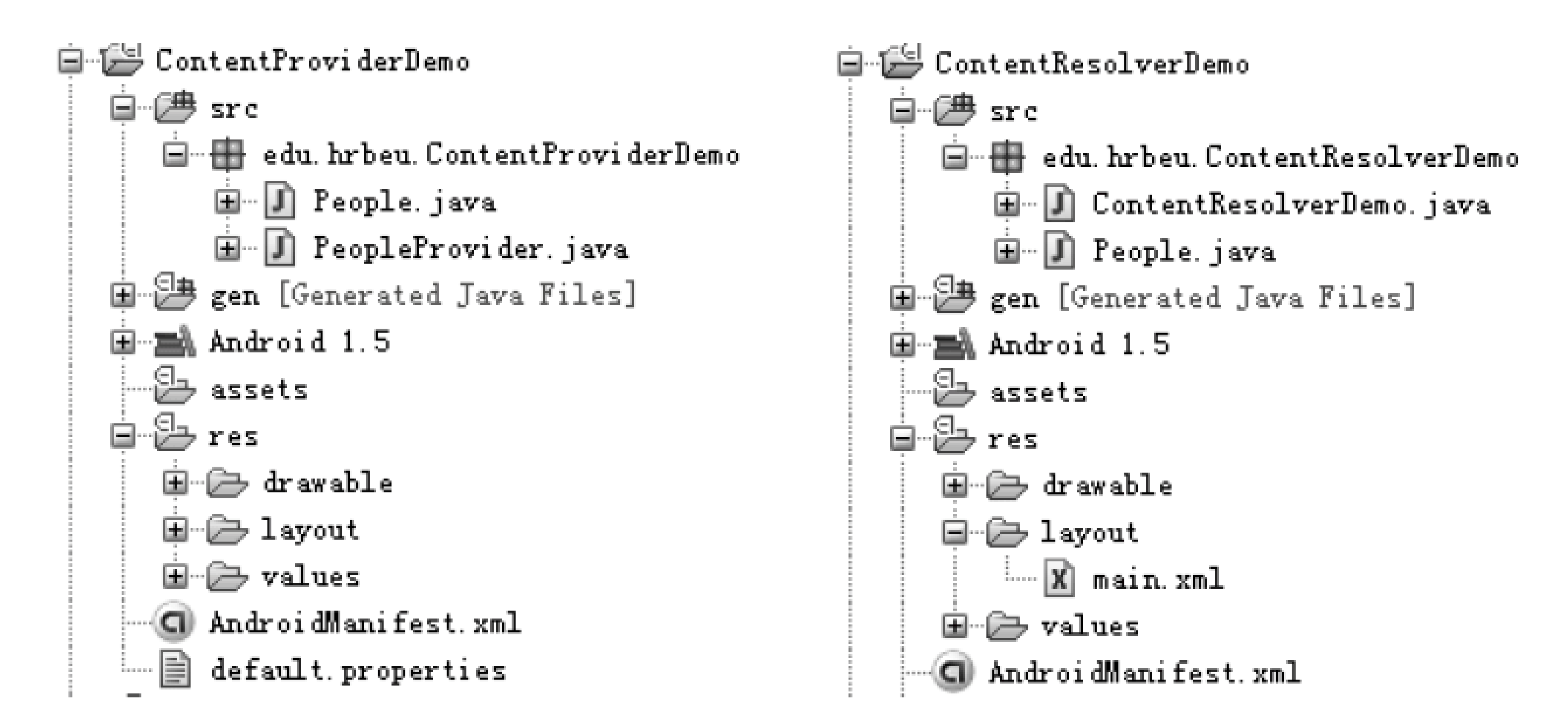



图 8.17 ContentProviderDemo 和 ContentResolverDemo 文件结构

下面分别给出 People.java、PeopleProvider.java 和 ContentResolverDemo.java 的完整代码，最后分别给出 ContentProviderDemo 示例和 ContentResolverDemo 示例的 AndroidManifest.xml 文件内容。

People.java 文件的完整代码如下：

```
package edu.hrbeu.ContentResolverDemo;
import android.net.Uri;

public class People{

    public static final String MIME_DIR_PREFIX="vnd.android.cursor.dir";
    public static final String MIME_ITEM_PREFIX="vnd.android.cursor.item";
    public static final String MINE_ITEM="vnd.hrbeu.people";

    public static final String MINE_TYPE_SINGLE=MIME_ITEM_PREFIX+"/"+MINE_ITEM;
    public static final String MINE_TYPE_MULTIPLE=MIME_DIR_PREFIX+"/"+MINE_ITEM;

    public static final String AUTHORITY="edu.hrbeu.peopleprovider";
    public static final String PATH_SINGLE="people/#";
    public static final String PATH_MULTIPLE="people";
    public static final String CONTENT_URI_STRING="content://"+AUTHORITY+"/"+PATH_MULTIPLE;
    public static final Uri  CONTENT_URI=Uri.parse(CONTENT_URI_STRING);

    public static final String KEY_ID="_id";
    public static final String KEY_NAME="name";
    public static final String KEY_AGE="age";
    public static final String KEY_HEIGHT="height";
}
```

PeopleProvider.java 文件的完整代码如下：

```
package edu.hrbeu.ContentProviderDemo;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.net.Uri;

```

```
public class PeopleProvider extends ContentProvider{

    private static final String DB_NAME= "people.db";
    private static final String DB_TABLE= "peopleinfo";
    private static final int DB_VERSION= 1;

    private SQLiteDatabase db;
    private DBOpenHelper dbOpenHelper;

    private static final int MULTIPLE_PEOPLE= 1;
    private static final int SINGLE_PEOPLE= 2;
    private static final UriMatcher uriMatcher;

    static {
        uriMatcher= new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(People.AUTHORITY, People.PATH_MULTIPLE, MULTIPLE_PEOPLE);
        uriMatcher.addURI(People.AUTHORITY, People.PATH_SINGLE, SINGLE_PEOPLE);
    }

    @Override
    public String getType(Uri uri) {
        switch(uriMatcher.match(uri)){
            case MULTIPLE_PEOPLE:
                return People.MINE_TYPE_MULTIPLE;
            case SINGLE_PEOPLE:
                return People.MINE_TYPE_SINGLE;
            default:
                throw new IllegalArgumentException("Unkown uri:"+ uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int count= 0;
        switch(uriMatcher.match(uri)){
            case MULTIPLE_PEOPLE:
                count= db.delete(DB_TABLE, selection, selectionArgs);
                break;
            case SINGLE_PEOPLE:
                String segment= uri.getPathSegments().get(1);
                count= db.delete(DB_TABLE, People.KEY_ID+ "= "+ segment, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unsupported URI:"+ uri);
```

```
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        long id=db.insert(DB_TABLE, null, values);
        if ( id>0 ){
            Uri newUri=ContentUris.withAppendedId(People.CONTENT_URI, id);
            getContext().getContentResolver().notifyChange(newUri, null);
            return newUri;
        }
        throw new SQLException("Failed to insert row into "+uri);
    }

    @Override
    public boolean onCreate() {
        Context context=getContext();
        dbOpenHelper=new DBOpenHelper(context, DB_NAME, null, DB_VERSION);
        db=dbOpenHelper.getWritableDatabase();

        if (db==null)
            return false;
        else
            return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder qb=new SQLiteQueryBuilder();
        qb.setTables(DB_TABLE);
        switch(uriMatcher.match(uri)){
            case SINGLE_PEOPLE:
                qb.appendWhere(People.KEY_ID+"="+uri.getPathSegments().get(1));
                break;
            default:
                break;
        }
        Cursor cursor=qb.query(db,
                projection,
                selection,
                selectionArgs,
```

```
                        null,
                        null,
                        sortOrder);
        cursor.setNotificationUri(getContext().getContentResolver(), uri);
        return cursor;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                String[] selectionArgs) {
        int count;
        switch(uriMatcher.match(uri)){
            case MULTIPLE_PEOPLE:
                count=db.update(DB_TABLE, values, selection, selectionArgs);
                break;
            case SINGLE_PEOPLE:
                String segment=uri.getPathSegments().get(1);
                count= db.update(DB_TABLE, values, People.KEY_ID+"="+ segment, selectionArgs);
                break;
            default:
                throw new IllegalArgumentException("Unknow URI:"+uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }


      private static class DBOpenHelper extends SQLiteOpenHelper {

          public DBOpenHelper(Context context, String name, CursorFactory factory, int version) {
            super(context, name, factory, version);
          }

          private static final String DB_CREATE="create table "+
            DB_TABLE+" ("+People.KEY_ID+" integer primary key autoincrement, "+
            People.KEY_NAME+" text not null, "+People.KEY_AGE+" integer,"+People.KEY_HEIGHT+" float);";

          @Override
          public void onCreate(SQLiteDatabase _db) {
            _db.execSQL(DB_CREATE);
          }
```

```
            @Override
            public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
              _db.execSQL("DROP TABLE IF EXISTS "+ DB_TABLE);
              onCreate(_db);
            }
          }
}
```

ContentResolverDemo.java 文件的完整代码如下：

```
package edu.hrbeu.ContentResolverDemo;


import android.app.Activity;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class ContentResolverDemo extends Activity {

    private EditText nameText;
    private EditText ageText;
    private EditText heightText;
    private EditText idEntry;

    private TextView labelView;
    private TextView displayView;

    private ContentResolver resolver;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        nameText= (EditText)findViewById(R.id.name);
        ageText= (EditText)findViewById(R.id.age);
```

```
        heightText= (EditText)findViewById(R.id.height);
        idEntry= (EditText)findViewById(R.id.id_entry);

        labelView= (TextView)findViewById(R.id.label);
        displayView= (TextView)findViewById(R.id.display);



        Button addButton= (Button)findViewById(R.id.add);
        Button queryAllButton= (Button)findViewById(R.id.query_all);
        Button clearButton= (Button)findViewById(R.id.clear);
        Button deleteAllButton= (Button)findViewById(R.id.delete_all);

        Button queryButton= (Button)findViewById(R.id.query);
        Button deleteButton= (Button)findViewById(R.id.delete);
        Button updateButton= (Button)findViewById(R.id.update);


        addButton.setOnClickListener(addButtonListener);
        queryAllButton.setOnClickListener(queryAllButtonListener);
        clearButton.setOnClickListener(clearButtonListener);
        deleteAllButton.setOnClickListener(deleteAllButtonListener);

        queryButton.setOnClickListener(queryButtonListener);
        deleteButton.setOnClickListener(deleteButtonListener);
        updateButton.setOnClickListener(updateButtonListener);

        resolver= this.getContentResolver();

    }


    OnClickListener addButtonListener= new OnClickListener() {
        @Override
        public void onClick(View v) {
            ContentValues values= new ContentValues();

            values.put(People.KEY_NAME, nameText.getText().toString());
            values.put(People.KEY_AGE, Integer.parseInt(ageText.getText().toString()));
            values.put(People.KEY_HEIGHT, Float.parseFloat(heightText.getText().toString()));

            Uri newUri= resolver.insert(People.CONTENT_URI, values);
```

```
                labelView.setText("添加成功，URI:"+newUri);

            }
        };

        OnClickListener queryAllButtonListener=new OnClickListener() {
            @Override
            public void onClick(View v) {
                Cursor cursor=resolver.query(People.CONTENT_URI,
                        new String[] { People.KEY_ID, People.KEY_NAME, People.KEY_AGE, People.KEY_HEIGHT},
                        null, null, null);
                if (cursor==null){
                    labelView.setText("数据库中没有数据");
                    return;
                }
                labelView.setText("数据库："+String.valueOf(cursor.getCount())+"条记录");

                String msg="";
                if (cursor.moveToFirst()){
                    do{
                        msg+="ID："+cursor.getInt(cursor.getColumnIndex(People.KEY_ID))+",";
                        msg+="姓名："+cursor.getString(cursor.getColumnIndex(People.KEY_NAME))+",";
                        msg+="年龄："+cursor.getInt(cursor.getColumnIndex(People.KEY_AGE))+", ";
                        msg+="身高："+cursor.getFloat(cursor.getColumnIndex(People.KEY_HEIGHT))+"\n";
                    }while(cursor.moveToNext());
                }

                displayView.setText(msg);
            }
        };

        OnClickListener clearButtonListener=new OnClickListener() {

            @Override
            public void onClick(View v) {
                displayView.setText("");
            }
        };

```

```
OnClickListener deleteAllButtonListener= new OnClickListener() {
    @Override
    public void onClick(View v) {
        resolver.delete(People.CONTENT_URI, null, null);
        String msg= "数据全部删除";
        labelView.setText(msg);
    }
};

OnClickListener queryButtonListener= new OnClickListener() {
    @Override
    public void onClick(View v) {
        Uri uri= Uri.parse(People.CONTENT_URI_STRING+ "/"+ idEntry.getText().toString());
        Cursor cursor= resolver.query(uri,
                new String[] { People.KEY_ID, People.KEY_NAME, People.KEY_AGE, People.
                KEY_HEIGHT},
                null, null, null);
        if (cursor= = null){
            labelView.setText("数据库中没有数据");
            return;
        }

        String msg= "";
        if (cursor.moveToFirst()){
            msg+ = "ID: "+ cursor.getInt(cursor.getColumnIndex(People.KEY_ID))+ ",";
            msg+ = "姓名: "+ cursor.getString(cursor.getColumnIndex(People.KEY_NAME))+
            ",";
            msg+ = "年龄: "+ cursor.getInt(cursor.getColumnIndex(People.KEY_AGE))+ ", ";
            msg+ = "身高: "+ cursor.getFloat(cursor.getColumnIndex(People.KEY_HEIGHT))
            + "\n";
          }

        labelView.setText("数据库: ");
        displayView.setText(msg);
    }
};

OnClickListener deleteButtonListener= new OnClickListener() {
    @Override
    public void onClick(View v) {

        Uri uri= Uri.parse(People.CONTENT_URI_STRING+ "/"+ idEntry.getText().toString());
        int result= resolver.delete(uri, null, null);
        String msg= "删除 ID为"+ idEntry.getText().toString()+ "的数据"+ (result> 0? "成
```

```
                功":"失败");
                labelView.setText(msg);
            }
        };

        OnClickListener updateButtonListener=new OnClickListener() {
            @Override
            public void onClick(View v) {
                ContentValues values=new ContentValues();

                values.put(People.KEY_NAME, nameText.getText().toString());
                values.put(People.KEY_AGE, Integer.parseInt(ageText.getText()
                .toString()));
                values.put(People.KEY_HEIGHT, Float.parseFloat(heightText.getText().toString
                ()));

                Uri uri=Uri.parse(People.CONTENT_URI_STRING+"/"+idEntry.getText().toString());
                int result=resolver.update(uri, values, null, null);

                String msg="更新 ID为"+idEntry.getText().toString()+"的数据"+(result>0?"成
                功":"失败");
                labelView.setText(msg);
            }
        };
}
```

ContentProviderDemo 示例的 AndroidManifest.xml 文件内容：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="edu.hrbeu.ContentProviderDemo"
          android:versionCode="1"
          android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <provider android:name=".PeopleProvider"
            android:authorities="edu.hrbeu.peopleprovider"/>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

ContentResolverDemo 示例的 AndroidManifest.xml 文件内容：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="edu.hrbeu.ContentResolverDemo"
          android:versionCode="1"
          android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ContentResolverDemo"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

## 习　　题

1. 应用程序一般允许用户自己定义配置信息，如界面背景颜色、字体大小和字体颜色等，尝试使用 SharedPreferences 保存用户的自定义配置信息，并在程序启动时自动加载这些自定义的配置信息。

2. 尝试把第 1 题的用户自己定义配置信息，以 INI 文件的形式保存在内部存储器上。

3. 简述在嵌入式系统中使用 SQLite 数据库的优势。

4. 分别使用手动建库和代码建库的方式，创建名为 test.db 的数据库，并建立 staff 数据表，表内的属性值如表 8.8 所示。

表 8.8 staff 表属性值

| 属　　性 | 数 据 类 型 | 说　　明 |
|---|---|---|
| _id | integer | 主键 |
| name | text | 姓名 |
| sex | text | 性别 |
| department | text | 所在部门 |
| salary | float | 工资 |

5. 利用第 4 题所建立的数据库和 staff 表，为程序提供添加、删除和更新等功能，并尝试将表 8.9 中的数据添加到 staff 表中。

表 8.9 peopleinfo 表内容

| _id | name | sex | department | salary |
|---|---|---|---|---|
| 1 | Tom | male | computer | 5400 |
| 2 | Einstein | male | computer | 4800 |
| 3 | Lily | female | 1.68 | 5000 |
| 4 | Warner | male | | |
| 5 | Napoleon | male | | |

6. 建立一个 ContentProvider，用来共享第 4 题所建立的数据库。

# 第 9 章 chapter 9

# 位置服务与地图应用

位置服务和地图应用是发展最为迅速、有着大量潜在需求的领域，通过本章的学习可以让读者简单地了解位置服务和地图应用的概念、方法和技巧。读者可以使用 Google 提供的地图服务，构建提供位置服务的应用程序。

**本章学习目标：**

- 了解位置服务的概念
- 了解地图密钥的申请方法
- 掌握获取位置信息的方法
- 掌握 MapView 和 MapController 的使用方法
- 掌握 Google 地图覆盖层的使用方法

## 9.1 位 置 服 务

位置服务（Location Based Services，LBS），又称定位服务或基于位置的服务，融合了 GPS 定位、移动通信、导航等多种技术，提供了与空间位置相关的综合应用服务。位置服务首先在日本得到商业化的应用。2001 年 7 月，DoCoMo 发布了第一款具有三角定位功能的手持设备，2001 年 12 月，KDDI 发布第一款具有 GPS 功能的手机。近些年来，基于位置的服务发展更加迅速，涉及到商务、医疗、工作和生活等各个方面，为用户提供定位、追踪和敏感区域警告等一系列服务。

Android 平台支持提供位置服务的 API，在开发过程中主要用到 LocationManager 和 LocationProviders 对象。

LocationManager 可以用来获取当前的位置，追踪设备的移动路线，或设定敏感区域，在进入或离开敏感区域时设备会发出特定警报。LocationProviders 则是能够提供定位功能的组件集合，集合中的每种组件以不同的技术提供设备的当前位置，区别在于定位的精度、速度和成本等方面。

为了使开发的程序能够提供位置服务，首先需要获得 LocationManager 对象。获取 LocationManager 可以通过调用 android. app. Activity. getSystemService()函数实现，代码如下：

```
1    String serviceString= Context.LOCATION_SERVICE;
2    LocationManager LocationManager= (LocationManager)getSystemService(serviceString);
```

代码第 2 行的 getSystemService()函数，可以根据服务名称获取 Android 提供的系统级服务。代码第 1 行的 Context. LOCATION_SERVICE 指明获取的服务是位置服务。Android 支持的系统级服务如表 9.1 所示。

**表 9.1 Android 支持的系统级服务**

| Context 类的静态常量 | 值 | 返回对象 | 说明 |
|---|---|---|---|
| LOCATION_SERVICE | location | LocationManager | 控制位置等设备的更新 |
| WINDOW_SERVICE | window | WindowManager | 最顶层的窗口管理器 |
| LAYOUT_INFLATER_SERVICE | layout_inflater | LayoutInflater | 将 XML 资源实例化为 View |
| POWER_SERVICE | power | PowerManager | 电源管理 |
| ALARM_SERVICE | alarm | AlarmManager | 在指定时间接受 Intent |
| NOTIFICATION_SERVICE | notification | NotificationManager | 后台事件通知 |
| KEYGUARD_SERVICE | keyguard | KeyguardManager | 锁定或解锁键盘 |
| SEARCH_SERVICE | search | SearchManager | 访问系统的搜索服务 |
| VIBRATOR_SERVICE | vibrator | Vibrator | 访问支持振动的硬件 |
| CONNECTIVITY_SERVICE | connection | ConnectivityManager | 网络连接管理 |
| WIFI_SERVICE | wifi | WifiManager | Wi-Fi 连接管理 |
| INPUT_METHOD_SERVICE | input_method | InputMethodManager | 输入法管理 |

在获取到 LocationManager 后，还需要指定 LocationManager 的定位方法，然后才能够调用 LocationManager. getLastKnowLocation() 方法获取当前位置。目前 LocationManager 支持的定位方法有两种，分别是使用 GPS 定位和使用网络定位。GPS 定位可以提供更加精确的位置信息，但定位速度和质量受到卫星数量和环境情况的影响；网络定位提供的位置信息精度较差，但速度较 GPS 定位快。LocationManager 支持定位方法参考表 9.2。

**表 9.2 LocationManager 支持定位方法**

| LocationManager 类的静态常量 | 值 | 说明 |
|---|---|---|
| GPS_PROVIDER | gps | 使用 GPS 定位，利用卫星提供精确的位置信息，需要 android. permissions. ACCESS_FINE_LOCATION 用户权限 |
| NETWORK_PROVIDER | network | 使用网络定位，利用基站或 Wi-Fi 提供近似的位置信息，需要具有如下权限：android. permission. ACCESS_COARSE_LOCATION 或 android. permission. ACCESS_FINE_LOCATION. |

在指定 LocationManager 的定位方法后，则可以调用 getLastKnowLocation()方法

获取当前的位置信息。以使用 GPS 定位为例，获取位置信息的代码如下：

```
1    String provider= LocationManager.GPS_PROVIDER;
2    Location location= locationManager.getLastKnownLocation(provider);
```

代码第 2 行返回的 Location 对象中，包含了可以确定位置的信息，如经度、纬度和速度等，用户可通过调用 Location 中的 getLatitude()和 getLonggitude()方法分别获取位置信息中的纬度和经度，示例代码如下：

```
double lat= location.getLatitude();
double lng= location.getLongitude();
```

在很多提供定位服务的应用程序中，不仅需要获取当前的位置信息，还需要监视位置的变化，在位置改变时调用特定的处理方法。LocationManager 提供了一种便捷、高效的位置监视方法 requestLocationUpdates()，可以根据位置的距离变化和时间间隔设定产生位置改变事件的条件，这样可以避免因微小的距离变化而产生大量的位置改变事件。LocationManager 中设定监听位置变化的代码如下：

```
locationManager.requestLocationUpdates(provider, 2000, 10, locationListener);
```

第 1 个参数是定位的方法，GPS 定位或网络定位；第 2 个参数是产生位置改变事件的时间间隔，单位为微秒；第 3 个参数是距离条件，单位是米；第 4 个参数是回调函数，是在满足条件后的位置改变事件的处理函数。上面的代码将产生位置改变事件的条件设定为距离改变 10 米，时间间隔为 2 秒。实现 locationListener 的代码如下：

```
1    LocationListener locationListener= new LocationListener(){
2        public void onLocationChanged(Location location) {
3        }
4        public void onProviderDisabled(String provider) {
5        }
6        public void onProviderEnabled(String provider) {
7        }
8        public void onStatusChanged(String provider, int status, Bundle extras) {
9        }
10   };
```

代码第 2 行的 onLocationChanged()在设备的位置改变时被调用；第 4 行的 onProviderDisabled()在用户禁用具有定位功能的硬件时被调用；第 6 行的 onProviderEnabled()在用户启用具有定位功能的硬件时被调用；第 8 行的 onStatusChanged()在提供定位功能的硬件的状态改变时被调用，如从不可获取位置信息状态到可以获取位置信息的状态，反之亦然。

最后，为了使 GPS 定位功能生效，还需要在 AndroidManifest. xml 文件中加入用户许可，代码如下：

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

CurrentLocationDemo 是一个提供位置服务的基本示例，提供了显示当前位置新的功能，并能够监视设备的位置变化。CurrentLocationDemo 的用户界面如图 9.1 所示。

位置服务一般都需要使用设备上的硬件，最理想的调试方式是将程序上传到物理设备上运行，但在没有物理设备的情况下，也可以使用 Android 模拟器提供的虚拟方式模拟设备的位置变化，调试具有位置服务的应用程序。首先打开 DDMS 中的模拟器控制，在 Location Controls 中的 Longitude 和 Latitude 部分输入设备当前的经度和纬度，然后单击 Send 按钮，就将虚拟的位置信息发送到 Android 模拟器中，如图 9.2 所示。

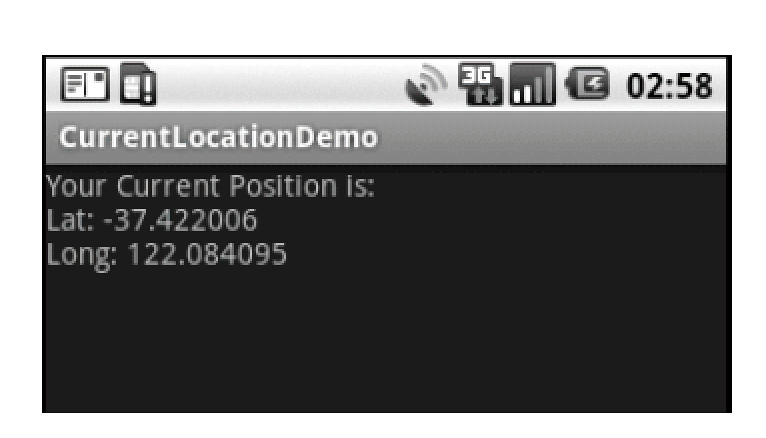



**图 9.1 CurrentLocationDemo 示例界面**

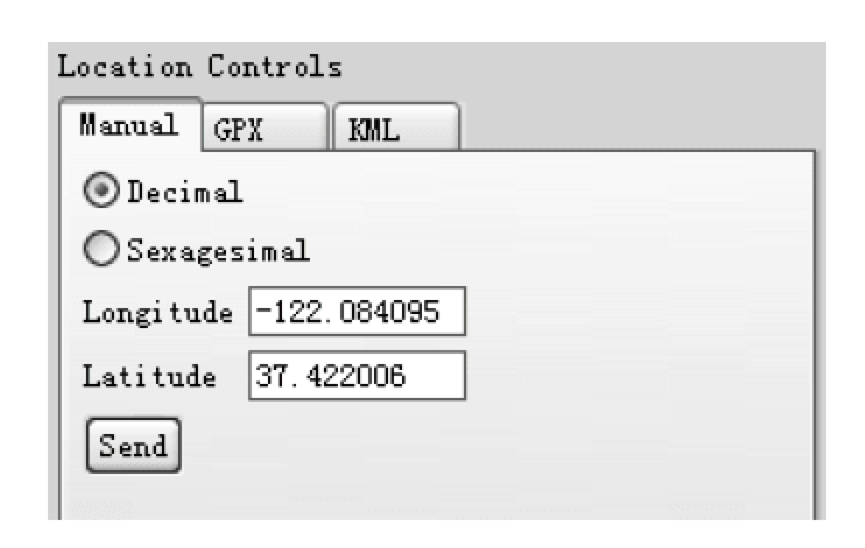



**图 9.2 模拟器控制器**

在程序运行过程中，可以在模拟器控制器中改变经度和纬度坐标值，程序在检测到位置的变化后，会将最新的位置信息显示在界面上。但笔者在 1.5 版本的 Android 模拟器中进行调试时，发现模拟器控制器只能成功地将虚拟坐标发送到模拟器中 2 次，超过 2 次后模拟器对新发送的虚拟坐标不再响应。

下面给出 CurrentLocationDemo 示例中 LocationBasedServiceDemo.java 文件的完整代码：

```
package edu.hrbeu.LocationBasedServiceDemo;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.TextView;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

public class LocationBasedServiceDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        String serviceString=Context.LOCATION_SERVICE;
    LocationManager locationManager= (LocationManager)getSystemService(serviceString);
        String provider=LocationManager.GPS_PROVIDER;
        Location location=locationManager.getLastKnownLocation(provider);
```

```
            getLocationInfo(location);
            locationManager.requestLocationUpdates(provider, 2000, 0, locationListener);
        }

        private void getLocationInfo(Location location){
            String latLongInfo;
            TextView locationText= (TextView)findViewById(R.id.txtshow);
            if (location != null){
                double lat= location.getLatitude();
                double lng= location.getLongitude();
                latLongInfo= "Lat: "+ lat+ "\nLong: "+ lng;
            }
            else{
                latLongInfo= "No location found";
            }
            locationText.setText("Your Current Position is:\n"+ latLongInfo);
        }

        private final LocationListener locationListener= new LocationListener(){
            @Override
            public void onLocationChanged(Location location) {
                getLocationInfo(location);
            }
            @Override
            public void onProviderDisabled(String provider) {
                getLocationInfo(null);
            }
            @Override
            public void onProviderEnabled(String provider) {
                getLocationInfo(null);
            }
            @Override
            public void onStatusChanged(String provider, int status, Bundle extras) {
            }
        };
    }
```

# 9.2 Google地图应用

## 9.2.1 申请地图密钥

为了在手机中更直观地显示地理信息，程序开发人员可以直接使用Google提供的

地图服务，实现地理信息的可视化开发。只要使用 MapView（com. google. android. maps. MapView）就可以将 Google 地图嵌入到 Android 应用程序中。但在使用 MapView 进行开发前，必须向 Google 申请一组经过验证的“地图密钥”（Map API Key），才能正常使用 Google 的地图服务。“地图密钥”是访问 Google 地图数据的密钥，无论是模拟器还是在真实设备中需要使用这个密钥。

注册“地图密钥”的第一步是申请一个 Google 账户，也就是 Gmail 电子邮箱，申请地址是 https://www. google. com/accounts/Login。

在得到 Google 账户之后，下一步工作是找到保存 Debug 证书的 keystore 的保存位置，并获取证书的 MD5 散列值。keystore 是一个密码保护的文件，用来存储 Android 提供的用于调试的证书，获取 MD5 散列值的主要目的是为下一步申请“地图密钥”做准备。获取证书的保存地址如图 9.3 所示，首先打开 Eclipse，通过 Windows→Preferences 打开配置窗体，在 Android → Build 栏中的 Default debug keystore 中可以找到。

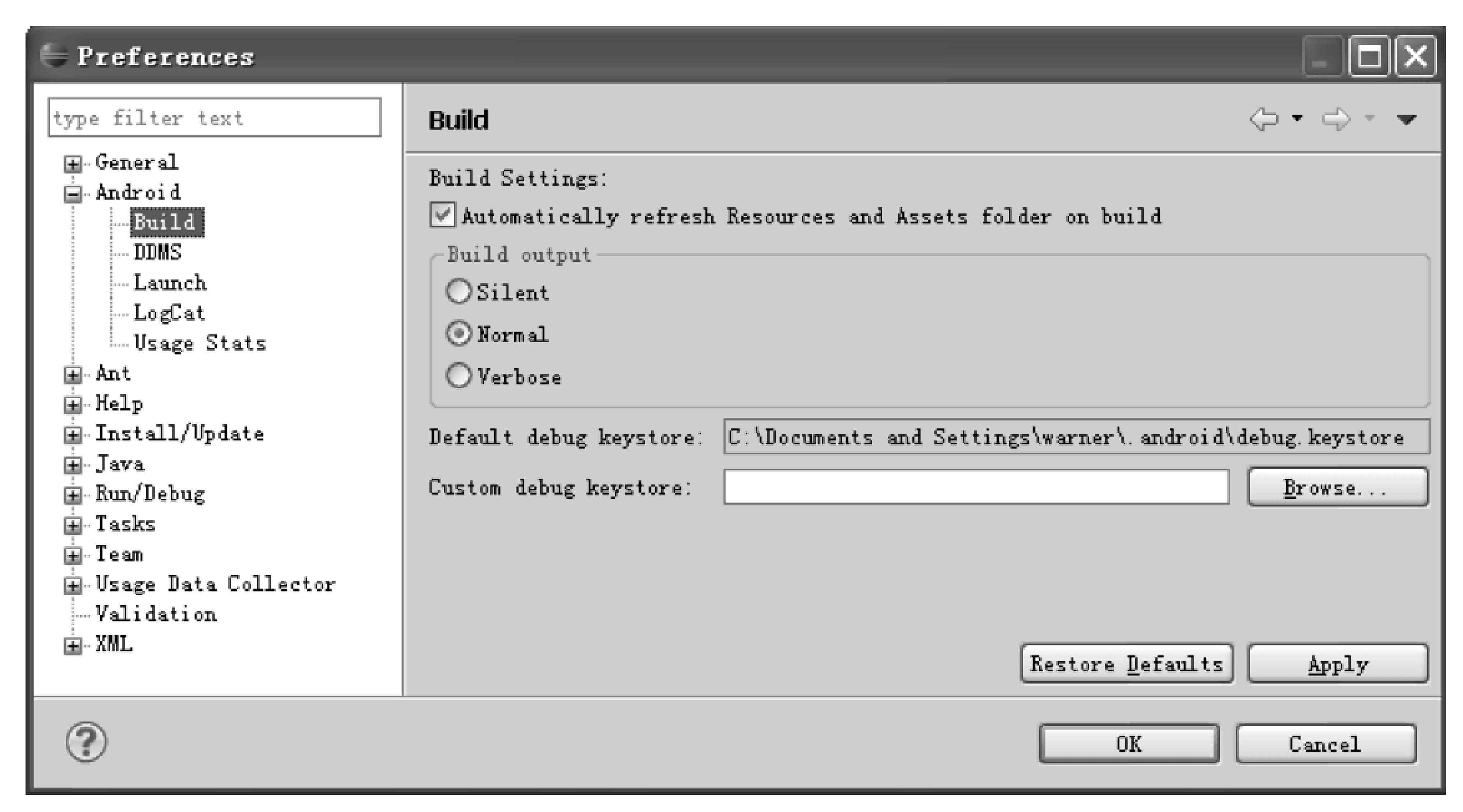



**图 9.3　命令运行结果**

为了获取 Debug 证书的 MD5 散列值，需要打开命令行工具 CMD，然后切换到 keystore 的目录，输入如下命令：

```
keytool -list -keystore debug.keystore
```

如果提示无法找到 keytool，可以将<Java SDK>/bin 的路径添加到系统的 PATH 变量中。在提示输入 keystore 密码时，输入默认密码 android，MD5 散列值将显示在最下方。如图 9.4 所示，笔者的 MD5 散列值为 68:76:89:C8:A4:24:61:F9:EA:F3:F7:70:CC:FD:C8:15。

申请“地图密钥”的最后一步是打开申请页面，输入 MD5 散列值。申请页面的地址是 http://code. google. com/intl/zh-CN/android/add-ons/google-apis/maps-api-signup. html，如图 9.5 所示。

输入 MD5 散列值后，单击 Generate API Key 按钮，将提示用户输入 Google 账户，正

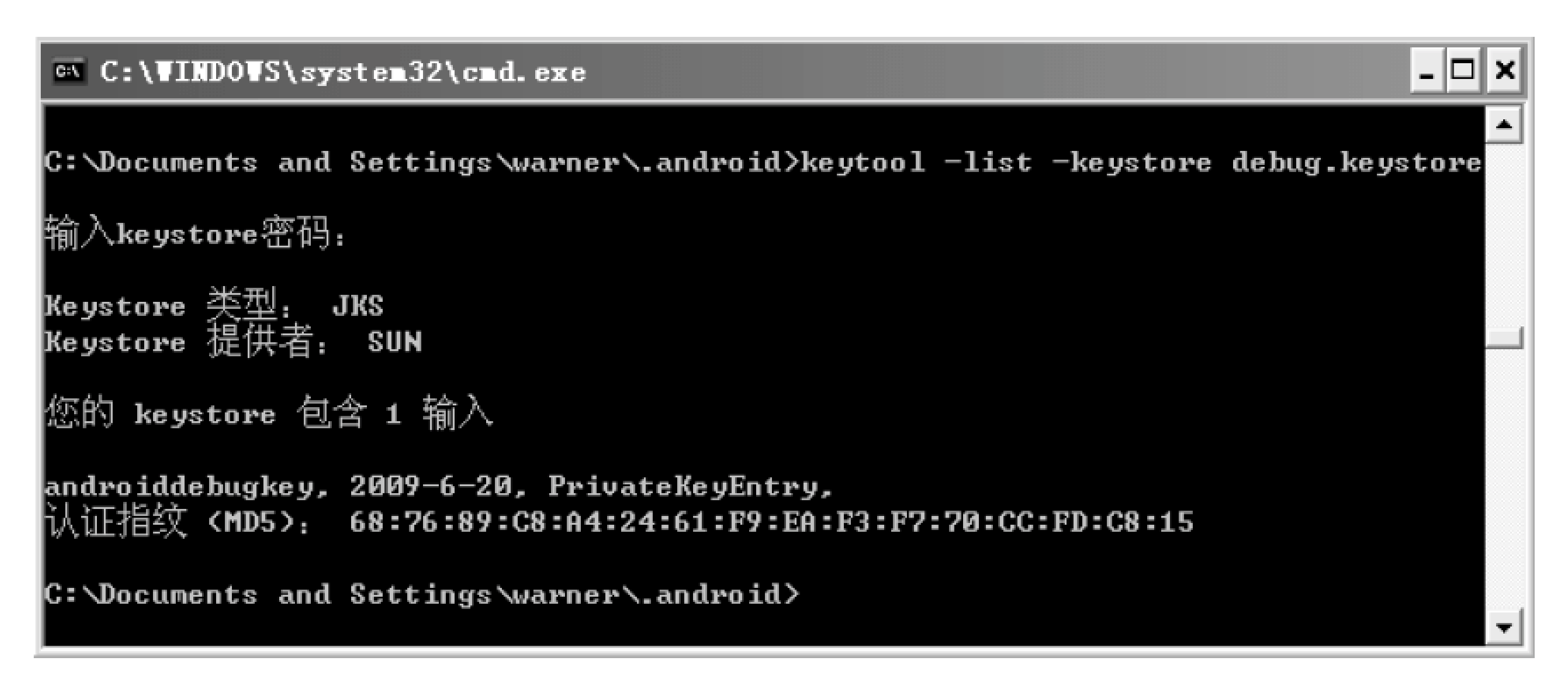



图 9.4　获取 keystore 的 MD5 散列值

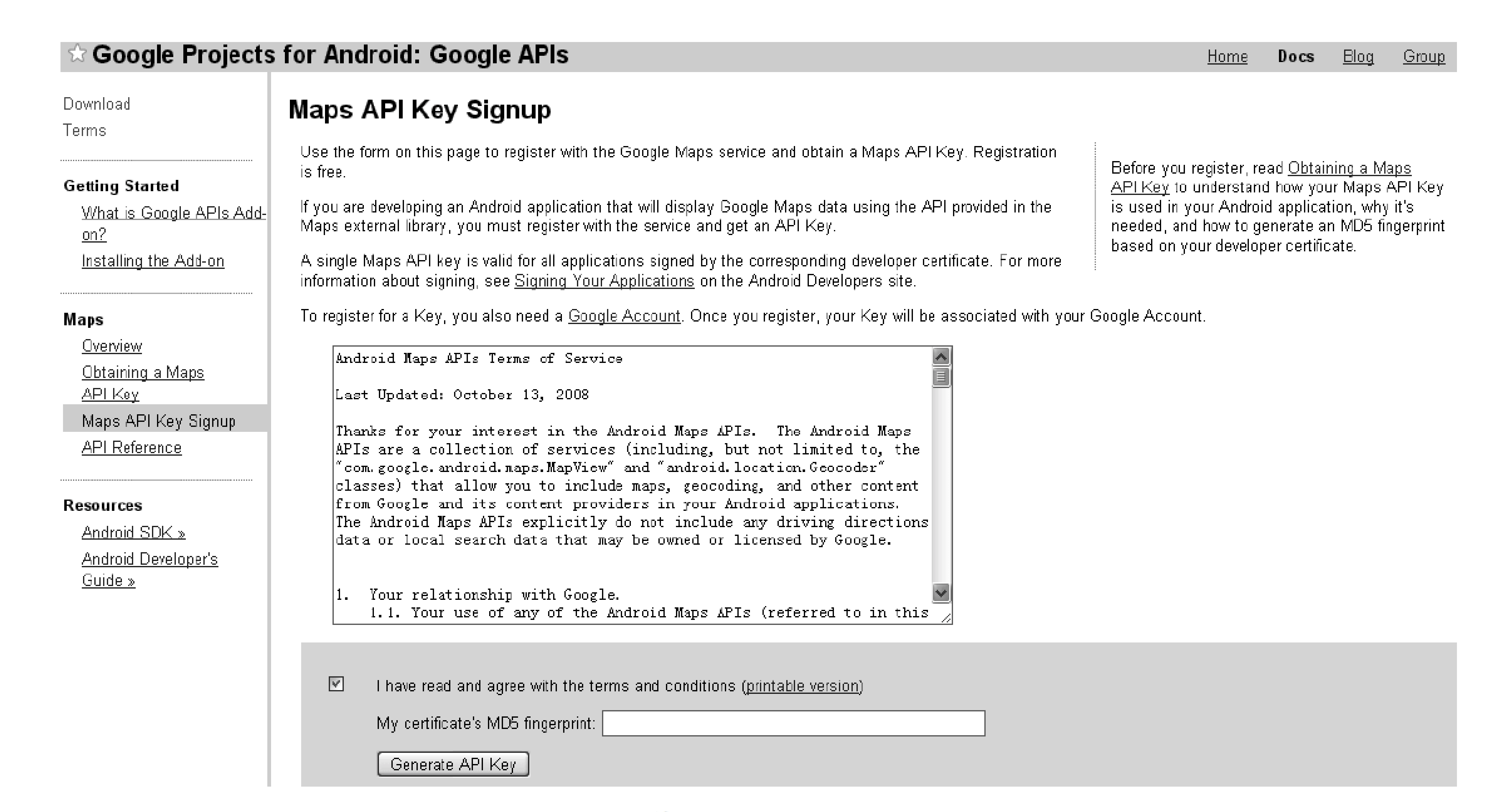



图 9.5　获取 Map API Key 页面

确输入 Google 账户后，将产生申请“地图密钥”的获取结果，如图 9.6 所示。

您的密钥是：

```
0mVK8GeO6WUz4S2F94z52CIGSSlvlTwnrE4DsiA
```

此密钥适用于所有使用以下指纹所对应证书进行验证的应用程序：

```
68:76:89:C8:A4:24:61:F9:EA:F3:F7:70:CC:FD:C8:15
```

下面是一个 xml 格式的示例，帮助您了解地图功能：

```
<com.google.android.maps.MapView
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:apiKey="0mVK8GeO6WUz4S2F94z52CIGSSlvlTwnrE4DsiA"
     />
```

图 9.6　“地图密钥”获取结果

笔者获取的“地图密钥”是0mVK8GeO6WUz4S2F94z52CIGSSlvlTwnrE4DsiA，在以后使用到MapView的时候都需要输入这个密钥。但需要注意的是，这个密钥读者并不能使用，读者需要根据自己的Debug证书的MD5散列值，重新到Google网站上申请一个用于调试程序的“地图密钥”。

## 9.2.2 使用Google地图

在申请到“地图密钥”后，下面考虑如何在Android系统中显示和控制Google地图。MapView的地图显示控件，可以设置不同的显示模式，例如卫星模式、街道模式或交通模式。而MapController则是MapView的控制器，可以控制MapView的显示中心和缩放级别等功能。

下面的内容以GoogleMapDemo为例，说明如何在Android系统中开发Google地图程序。这个示例将在程序内部设置一个坐标点，然后在程序启动时，使用MapView控件在地图上显示这个坐标点的位置。

因为普通的Android程序并不包含支持Google地图开发的扩展库，因此应在建立工程时将com. google. android. maps的扩展库添加到工程中，这样就可以使用Google地图的所有功能了。添加com. google. android. maps扩展库的方式是在创建工程时，在Build Target项中选择Google APIs，如图9.7所示。




**图9.7 引入Google地图扩展库**

创建工程后，修改/res/layout/main. xml文件，在布局中加入一个MapView控件，并设置刚获取的“地图密钥”。main. xml文件的完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>

    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
```

```
            android:enabled="true"
            android:clickable="true"
            android:apiKey="0mVK8GeO6WUz4S2F94z52CIGSSlvlTwnrE4DsiA"/>
</LinearLayout>
```

仅在布局中添加 MapView 控件，还不能够直接在程序中调用这个控件，还需要将程序本身设置成 MapActivity（com.google.android.maps.MapActivity），MapActivity 类负责处理显示 Google 地图所需的生命周期和后台服务管理。下面先给出整个 GoogleMapDemo.java 文件的完整代码：

```
package edu.hrbeu.GoogleMapDemo;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

import android.os.Bundle;

public class GoogleMapDemo extends MapActivity {
    private MapView mapView;
    private MapController mapController;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView= (MapView)findViewById(R.id.mapview);
        mapController=mapView.getController();

        Double lng=126.676530486 * 1E6;
        Double lat=45.7698895661 * 1E6;
        GeoPoint point=new GeoPoint(lat.intValue(), lng.intValue());

        mapController.setCenter(point);
        mapController.setZoom(11);
        mapController.animateTo(point);

        mapView.setSatellite(false);
    }

    @Override
    protected boolean isRouteDisplayed() {
```

```
35              // TODO Auto-generated method stub
36              return false;
37          }
38      }
```

代码第 20 行获取了 MapController;在第 26 行设置 MapView 的“显示中点”;在第 27 行设置放大层级;在第 28 行将 MapView 显示区域的中心移动到第 26 行设置的“显示中心”;代码第 22 行和第 23 行设定的经度为 126.676530486 * 1E6、纬度为 45.7698895661 * 1E6 的地理坐标点;在代码第 26 行,没有直接使用这个坐标的,而是将其转化为 GeoPoint 再使用;代码 30 行是设定 MapView 的地图显示模式是否为卫星模式,设置 true 则为卫星模式,设置 false 则为普通模式;代码第 34 行 isRouteDisplayed()方法,是用来统计程序是否在 Google 地图中显示路径信息,默认为不显示。

运行前还需要在 AndroidManifest.xml 文件中添加允许访问互联网的许可,原因是获取 Google 地图是需要使用互联网的。AndroidManifest.xml 文件的完整代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hrbeu.GoogleMapDemo"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".GoogleMapDemo"
                android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.
                LAUNCHER" />
            </intent-filter>
        </activity>
        <uses-library android:name="com.google.android.maps"></uses-library>
    </application>

    <uses-sdk android:minSdkVersion="3" />
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

最后,程序运行时需要连接互联网,运行结果如图 9.8 所示。

### 9.2.3 使用 Overlay

在目前的很多基于地图的应用中,都需要在地图上显示信息或绘制图形。通过在 MapView 上添加覆盖层,可以在指定的位置加添加注解、绘制图像或处理进行鼠标事件等。Google 地图上可以加入多个覆盖层,所有覆盖层均在地图图层之上,每个覆盖层均可以对用户的点击事件做出响应。







(a) 地图模式　　　　(b) 卫星模式

**图 9.8　GoogleMapDemo 示例运行结果**

创建覆盖层继承 Overlay 类的子类,并通过重载 draw()方法为指定位置添加注解,重载 onTap()方法处理用户的点击操作。下面的代码是创建 Overlay 的最小代码集合:

```
1   public class TextOverlay extends Overlay {
2       @Override
3       public void draw(Canvas canvas, MapView mapView, boolean shadow) {
4           if (shadow==false){
5
6           }
7           else{
8           }
9           super.draw(canvas, mapView, shadow);
10      }
11
12      @Override
13      public boolean onTap(GeoPoint p, MapView mapView) {
14          return false;
15      }
16  }
```

在代码第 3 行的 draw()方法中,shadow 变量是用来区分绘制图层的,false 表示在覆盖层上进行绘制,而 true 则表示在隐藏层上进行绘制。代码的第 14 行是 onTap()方法的返回值,返回 false 表示覆盖层不处理点击事件,返回 true 则表示已经处理了点击事件。

在覆盖层绘制图形或文字需要使用“画布”(Canvas)来实现，绘制的位置是屏幕坐标，这就需要将地图上的物理坐标与屏幕坐标进行转换。Projection 类提供了物理坐标和屏幕坐标的转换功能，可在经度和纬度表示的 GeoPoint 点和屏幕上 Point 点进行转换。其中，toPixels()方法将物理坐标转换为屏幕坐标，fromPixels()方法将屏幕坐标转换为物理坐标，两个方法的具体使用方法可以参考下面的代码：

```
Projection projection=mapView.getProjection();

projection.toPixels(geoPoint, point);
projection.fromPixels(point.x, point.y);
```

下面的内容以 MapOverlayDemo 示例，说明如何在 Google 地图上添加覆盖层，并在预订的物理坐标上显示提示信息。MapOverlayDemo 示例的运行结果如图 9.9 所示。

TextOverlay 类是 MapOverlayDemo 示例的覆盖层，主要重载了 draw()方法，在指定的物理坐标上绘制了标记点和提示文字。TextOverlay.java 文件的核心代码如下：

```
public class TextOverlay extends Overlay {
    private final int mRadius=5;

    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {
        Projection projection=mapView.getProjection();

        if (shadow==false){
            Double lng=126.676530486*1E6;
            Double lat=45.7698895661*1E6;
            GeoPoint geoPoint=new GeoPoint(lat.intValue(), lng.intValue());

            Point point=new Point();
            projection.toPixels(geoPoint, point);

            RectF oval=new RectF(point.x-mRadius, point.y-mRadius,
                                 point.x+mRadius, point.y+mRadius);

            Paint paint=new Paint();
            paint.setARGB(250, 250, 0, 0);
            paint.setAntiAlias(true);
            paint.setFakeBoldText(true);

            canvas.drawOval(oval, paint);
            canvas.drawText("标记点", point.x+2*mRadius, point.y, paint);
```

图 9.9 MapOverlayDemo 运行结果

```
25            }
26            super.draw(canvas, mapView, shadow);
27        }
28
29        @Override
30        public boolean onTap(GeoPoint p, MapView mapView) {
31            return false;
32        }
33    }
```

代码第 2 行定义了绘制半径变量 mRadius，供定义绘制范围使用；代码第 14 行使用 Projection 完成了从物理坐标到屏幕坐标的转换；第 19 行设置了绘制颜色；第 20 行开启了平滑设置，防止文字出现锯齿；代码第 23 行绘制了圆形的标记点，标记点的大小以代码第 16 行设定的 oval 为准；代码第 24 行绘制了提示文字，第 2 个和第 3 个参数是绘制屏幕的 x 坐标和 y 坐标。

建立了覆盖层后，还需要把覆盖层添加到 MapView 上。MapOverlayDemo.java 的核心代码如下：

```
public class MapOverlayDemo extends MapActivity {
    private MapView mapView;
    private MapController mapController;
    private TextOverlay textOverlay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView= (MapView)findViewById(R.id.mapview);
        mapController=mapView.getController();

        Double lng=126.676530486 * 1E6;
        Double lat=45.7698895661 * 1E6;
        GeoPoint point=new GeoPoint(lat.intValue(), lng.intValue());

        mapController.setCenter(point);
        mapController.setZoom(11);
        mapController.animateTo(point);

        textOverlay=new TextOverlay();
        List<Overlay> overlays=mapView.getOverlays();
        overlays.add(textOverlay);
    }

```

```
27        @Override
28        protected boolean isRouteDisplayed() {
29            return false;
30        }
31    }
```

代码第22行实例化了TextOverlay对象；在第23行通过getOverlays()方法，获取MapView已有的覆盖层；在第24行使用add()方法将TextOverlay对象添加到MapView中。

## 习　题

1. 讨论位置服务和地图应用的发展前景。

2. 编程实现轨迹追踪软件。每间隔60秒，同时距离移动大于1米的情况下，记录一次位置信息，在Google地图上绘制600秒的行动轨迹。

# 第 10 章 chapter 10

# Android NDK 开发

Android NDK 的出现使 Android 平台能够使用 C/C++等非托管代码开发应用程序,对 Android 平台的普及具有深远的意义。通过本章的学习可以让读者初步了解 Android NDK 的开发方法。

**本章学习目标:**

- 了解 Android NDK 的用途
- 掌握 Android NDK 编译环境的安装与配置方法
- 掌握 Android NDK 的开发方法

## 10.1 NDK 简 介

Android NDK(Android Native Development Kit)是一系列的开发工具,允许程序开发人员在 Android 应用程序中嵌入 C/C++语言编写的非托管代码。一般情况下,Android 应用程序主要使用 Java 语言进行开发,编译后产生的托管代码在 Dalvik 虚拟机上运行。但在一些需要较高执行效率的地方,程序开发人员希望能够使用非托管代码,以提高 Android 应用程序的核心部分的运行速度。不仅如此,程序开发人员还希望能够直接使用完善、成熟的 C/C++源代码,以提高 Android 应用程序的开发速度。Android NDK 的出现,不仅解决了核心模块使用托管语言开发执行效率低下的问题,还允许程序开发人员直接使用 C/C++源代码,极大地提高了 Android 应用程序开发的灵活性。

当然,在使用 Android NDK 过程中不能仅看到它所带来的好处,程序开发人员必须清楚认识到 Android NDK 的不足。首先,虽然使用 C/C++语言开发 Android 程序会提高运行效率,但同时也会增加程序的复杂性,增加了程序调试的难度,因此程序开发人员需要在运行效率和复杂性之间做个选择。其次,程序开发人员需要认识到 CPU 运算量大和内存消耗较少的部分(例如信号处理或物理仿真)使用非托管代码运行效率最高,因此并不是所有的核心部分都适合使用 C/C++语言编写。

目前 Android NDK 的版本是 1.5,集成了交叉编译器,支持 ARMv5TE 处理器指令集、JNI 接口和一些稳定的库文件,但仅支持 Android SDK 1.5 版本,因此 1.0 和 1.1 版本的应用程序不能够使用 Android NDK。Android NDK 提供一系列的说明文档、示例代码和开发工具,指导程序开发人员使用 C/C++语言进行库文件开发,并提供便捷工具

将库文件打包到 apk 文件中。

## 10.2 NDK 编译环境

Android NDK 编译环境支持 Windows XP、Linux 和 Mac OS，本书仅介绍 Windows 系统的编译环境配置方法。

第一步，在 Google 的官方网站下载 Android NDK 的安装包，下载地址是 http://developer.android.com/sdk/ndk/1.5_r1/index.html，打开下载页面后选择的下载文件为 android-ndk-1.5_r1-windows.zip，如图 10.1 所示。将下载的 ZIP 文件解压缩到用户的 Android 开发目录中，作者将 Android NDK 解压到 E:\Android 目录中，ZIP 文件中包含一层目录，因此 Android NDK 的最终路径为 E:\Android\android-ndk-1.5_r1。

**Download Android 1.5 NDK, Release 1**

*June 2009*

The Android NDK is a companion tool to the Android SDK that lets Android application developers build performance-critical portions of their apps in native code. It is designed for use *only* in conjunction with the Android SDK, so if you have not already installed the Android 1.5 SDK, please do so before downloading the NDK. Also, please read What is the Android NDK? to get an understanding of what the NDK offers and whether it will be useful to you.

Select the download package that is appropriate for your development computer.

| Platform | Package | Size | MD5 Checksum |
|---|---|---|---|
| Windows | android-ndk-1.5_r1-windows.zip | 22500667 bytes | e5c53915903d8b81f3e2ea422e2e2717 |
| Mac OS X (intel) | android-ndk-1.5_r1-darwin-x86.zip | 17215303 bytes | 1931f0e182798a4c98924fd87380b5b8 |
| Linux 32/64-bit (x86) | android-ndk-1.5_r1-linux-x86.zip | 16025885 bytes | 80a4e14704ca84c21bf1824cb25fbd8b |

**图 10.1 Android NDK 下载页面**

第二步是下载并安装 Cygwin。目前，Android NDK 还不支持在 Windows 系统下直接进行交叉编译，因此需要在 Windows 系统下安装一个 Linux 的模拟器环境 Cygwin，完成 C/C++代码的交叉编译工作。Android NDK 要求 GNU Make 的版本高于或等于 3.18，之前的版本并没有经过测试，因此需要安装较新版本的 Cygwin。Cygwin 的最新版本可以到官方网站 http://www.cygwin.com 下载，也可以到中文的映像网站 http://www.cygwin.cn 下载。在 Cygwin 的安装过程中，需要将 Devel 下的 gcc 和 make 的相关选项选上，如图 10.2 所示，否则 Cygwin 将无法编译 C/C++代码文件。

第三步是配置 Cygwin 的 NDK 开发环境。在默认情况下，Cygwin 安装在 C 盘的根目录下，修改 C:\cygwin\home\username。bash_profile 文件，username 会根据用户使用的用户名称而变化。在.bash_profile 文件的结尾处添加如下代码：

```
ANDROID_NDK_ROOT=/cygdrive/e/android/android-ndk-1.5_r1
export ANDROID_NDK_ROOT
```

上面的代码说明了 Android NDK 所在的目录，目录是 e 盘 android/android-ndk-1.5_r1。如果用户的 Android NDK 安装在 C 盘的 TestAndroid/android-ndk-1.5_r1 中，则上面的代码则应该为：

```
ANDROID_NDK_ROOT=/cygdrive/c/TestAndroid/android-ndk-1.5_r1
export ANDROID_NDK_ROOT
```

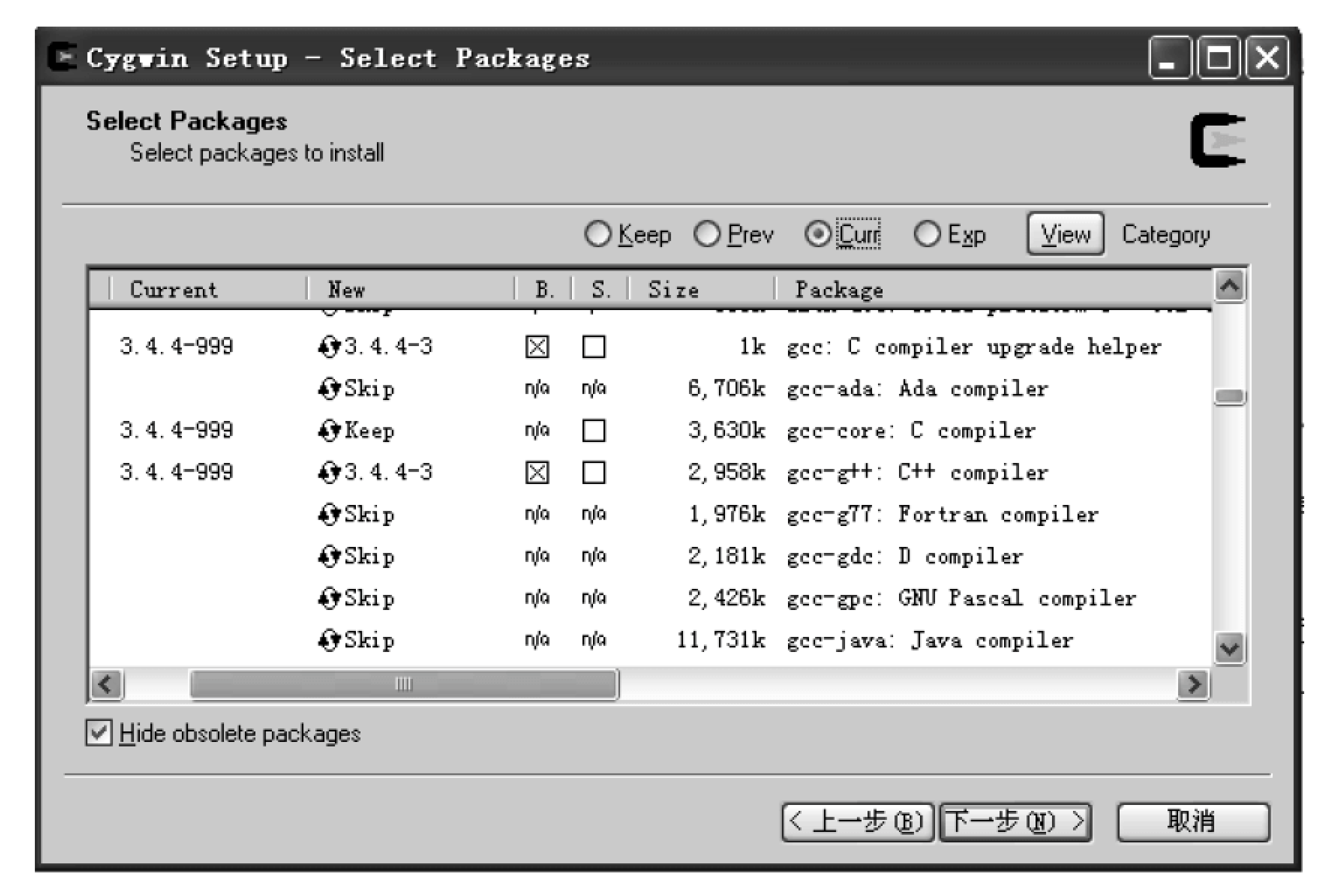



图 10.2　Cygwin 安装选项

最后一步是测试开发环境是否可以正常工作。首先启动 Cygwin，然后切换到<Android NDK>/build 目录中，运行 host-setup.sh 文件。如果出现图 10.3 所示的运行结果，说明 Android NDK 的开发环境已经可以正常工作了。

```
warner@jorin1 /cygdrive/e/Android/android-ndk-1.5_r1/build
$ ./host-setup.sh
Detecting host toolchain.

CC          : compiler check ok (gcc)
LD          : linker check ok (gcc)
CXX         : C++ compiler check ok (g++)
Generate    : out/host/config.mk
Toolchain   : Checking for arm-eabi-4.2.1 prebuilt binaries

Host setup complete. Please read docs/OVERVIEW.TXT if you don't know what to do.


warner@jorin1 /cygdrive/e/Android/android-ndk-1.5_r1/build
$ _
```

图 10.3　运行 host-setup.sh 文件

到此为止，Android NDK 的编译环境已经安装配置完毕，下面介绍 Android NDK 的目录结构，以及各个目录或文件的作用。在 android-ndk-1.5_r1 目录中，包含 5 个子目录和 2 个文件，结构如下所示：

```
(+)apps
   (+)hello-jni
   (+)tow-libs
(+)build
(+)docs
(+)out
   (-).gitignore
```

```
(+)sources
  (+)samples
      (+)hello-jni
      (+)tow-libs
      (-)Android.mk
(-)GNUmakefile
(-)README.TXT
```

其中，apps目录是Android工程的保存目录，子目录hello-jni和tow-libs是NDK自带的两个示例目录。build目录保存了交叉编译工具、编译脚本和配置文件。docs目录是帮助文档的保存目录，帮助文档的名称和用途参考表10.1。out目录是交叉编译的输出目录，保存输出的so文件。sources目录是C/C++源代码文件的保存目录，其下的hello-jni和tow-libs子目录，分别保存了NDK自带示例所需要的C/C++源代码文件。GNUmakefile文件和README.TXT文件分别是make工具的配置文件和NDK的说明文件。

**表10.1 docs目录中的帮助文件说明**

| 文件名 | 说明 |
|---|---|
| INSTALL.TXT | NDK的安装与配置说明文档 |
| OVERVIEW.TXT | NDK的用途和使用范围的说明文档 |
| ANDROID-MK.TXT | Android.mk文件的说明文档，Android.mk文件定义了需要编译的C/C++源代码 |
| APPLICATION-MK.TXT | Application.mk文件的说明文档，Application.mk文件定义了Android工程需要调用的C/C++源代码 |
| HOWTO.TXT | 关于NDK开发的一般性信息 |
| SYSTEM-ISSUES.TXT | 使用NDK开发时所需要注意的问题 |
| STABLE-APIS.TXT | NDK头文件所支持的稳定的API类表 |

Android NDK自带两个示例hello-jni和tow-libs。hello-jni是一个非常简单的例子，非托管代码实现了一个可以返回字符串的共享库，Android工程调用这个共享库获取字符串，然后显示在用户界面上。tow-libs是稍微复杂一些的例子，使用非托管代码实现了一个数学运算的共享库，Android工程动态加载这个共享库，并调用其中的函数，函数功能是通过使用静态库实现的。

## 10.3 NDK开发示例

在进行NDK开发时，一般需要同时建立Android工程和C/C++工程，然后使用NDK编译C/C++工程，形成可以被调用的共享库，最后共享库文件会被拷贝到Android工程中，并被直接打包到apk文件中。

后面的内容将 AndroidNdkDemo 示例说明如何进行 Android NDK 开发。AndroidNdkDemo 是一个进行加法运算的示例，程序会随机产生两个整数，然后调用 C 语言开发的共享库对这两个整数进行加法运算，最后将运算结果显示在用户界面上。AndroidNdkDemo 示例的界面如图 10.4 所示。

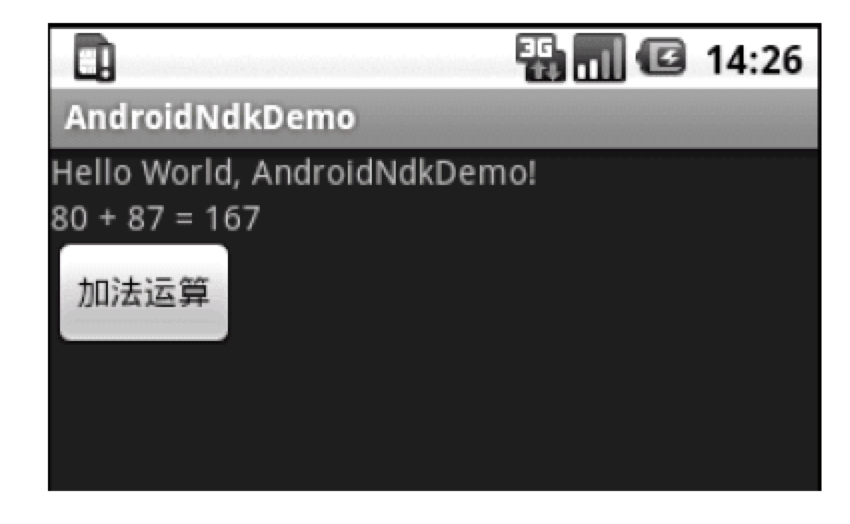



**图 10.4 AndroidNdkDemo 示例界面**

进行 Android NDK 开发一般要经过如下的步骤：

（1）建立 Application.mk 文件。

（2）建立 Android 工程。

（3）建立 Android.mk 文件。

（4）建立 C 源代码文件。

（5）编译共享库模块。

（6）运行 Android 程序。

下面的内容将详细介绍 Android NDK 开发的每一个步骤。



### 1. 建立 Application.mk 文件

进行 Android NDK 开发前，首先需要在 apps 目录中建立应用程序目录，AndroidNdkDemo 示例的应用程序目录为 ndk-demo，然后在 ndk-demo 目录中建立一个空目录 project，这个目录以后会用来存放 Android 工程。最后在 ndk-demo 目录中建立一个名为 Application.mk 的文件，用来描述 Android 工程将调用的共享库。AndroidNdkDemo 示例的目录结构如下所示：

```
(+)apps
   (+)ndk-demo
       (+)project
       (-)Application.mk
```

在进行 NDK 开发时，在应用程序目录中一定要有 Application.mk 文件，用来声明 Android 工程需要调用的非托管模块（如静态库或共享库）。AndroidNdkDemo 示例的 Application.mk 的代码如下：

```
1    APP_PROJECT_PATH := $ (call my-dir)/project
2    APP_MODULES          := add-module
```

在上面的代码中，第 1 行的变量 APP_PROJECT_PATH 表示 Android 工程所在的目录，在生产共享库文件后，APK 生产工具将自动将共享库文件拷贝到＜app＞\libs\armeabi 目录中，本示例将共享库文件拷贝到 apps\ndk-demo\project\libs\armeabi 目录中。第 2 行代码中的变量 APP_MODULES 表示 Android 工程需要调用的非托管模块，如果存在多个非托管模块，使用空格进行分隔。本示例调用的非托管模块为 add-module，对应在后面涉及的 Android.mk 文件。Application.mk 的所有变量说明可以参考表 10.2。

**表 10.2 Application.mk 的变量说明**

| 变 量 | 强制使用 | 说 明 |
|---|---|---|
| APP_PROJECT_PATH | 是 | Android 工程所在的目录 |
| APP_MODULES | 是 | Android 工程需要调用的非托管模块 |
| APP_OPTIM | 否 | 指定优化等级,包含两个等级 debug 和 release,release 是默认设置。debug 生产非优化代码,更加易于调试 |
| APP_CFLAGS | 否 | 编译 C 代码时所传递的编译器标志 |
| APP_CXXFLAGS | 否 | 编译 C++代码时所传递的编译器标志 |
| APP_CPPFLAGS | 否 | 编译 C/C++代码时所传递的编译器标志 |

### 2. 建立 Android 工程

下面来说明如何在 project 目录中建立 Android 工程。在建立 Android 工程时,需要取消复选框 Use default location,并指定预先建立的 project 文件夹作为工程文件夹,如图 10.5 所示。

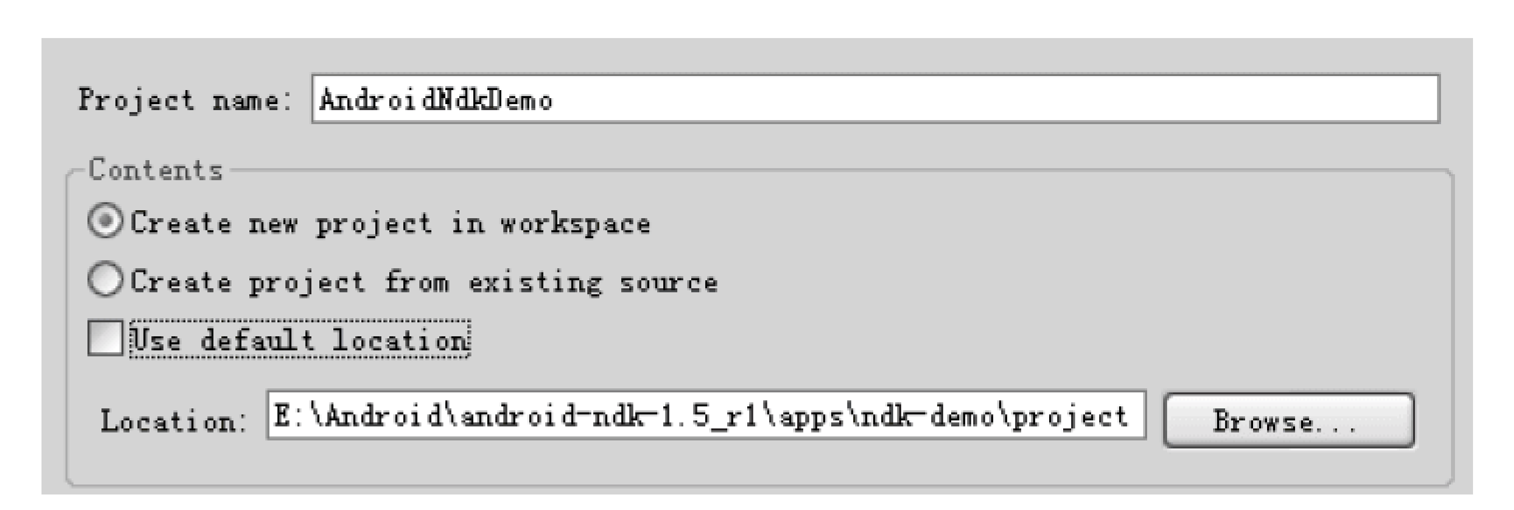



**图 10.5 建立 AndroidNdkDemo 工程**

在建立 AndroidNdkDemo 工程后,修改 main.xml 文件,添加一个 id 为 display 的 TextView 和一个 id 为 add_btn 的 Button 按钮。程序中的生产随机数和调用的代码在 AndroidNdkDemo.java 文件中,下面是 AndroidNdkDemo.java 文件的核心代码:

```
public class AndroidNdkDemo extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final TextView displayLable= (TextView)findViewById(R.id.display);
        Button btn= (Button)findViewById(R.id.add_btn);
        btn.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                double randomDouble=Math.random();
                long x=Math.round(randomDouble * 100);
                randomDouble=Math.random();
                long y=Math.round(randomDouble * 100);
```

```
15
16                      //System.loadLibrary("add-module");
17                      long   z=add(x, y);
18                      String msg=x+"+"+y+"="+z;
19                      displayLable.setText(msg);
20                  }
21              });
22          }
23          //public native long add(long   x, long   y);
24
25          public long add(long   x, long   y){
26              return x+y;
27          }
28      }
```

上面的代码有一个 NDK 开发的小技巧，在开发 C/C++的共享库前，可以使用具有相同和相近功能的 Java 函数进行替代。在代码第 17 行本应该调用共享库的 add()函数，但为了便于开发和调试，在代码第 25 行到第 27 行，使用 Java 代码开发了一个功能相同的 add()函数，这样即使在没有完成 C/C++的共享库开发前，也可以对这个 Android 工程进行界面部分的调试。第 16 行和第 23 行注释掉的代码，就是在 C/C++的共享库开发完毕后需要使用的代码，其中第 16 行是动态加载共享库的代码，加载的共享库名称为 add-module；第 23 行用来声明共享库的函数，使用 C/C++开发的共享库必须有同名的函数。在共享库开发完毕后，取消第 16 行和第 23 行代码的注释，并注释掉第 25 行到第 27 行代码，这样程序就可以正常调用共享库内的函数进行加法运算了。

### 3. 建立 Android.mk 文件

建立 C/C++源代码文件前，首先需要在 sources 目录中建立模块目录，AndroidNdkDemo 示例的模块目录为 add-module，这个模块目录的名称与 Application.mk 文件中声明的模块名称相同。add-module 目录中包含两个文件，Android.mk 和 add-module.c，目录结构如下所示：

```
(+)sources
    (+)add-module
        (-)Android.mk
        (-)add-module.c
```

Android.mk 是为 NKD 编译系统准备的脚本文件，用来描述模块需要编译 C/C++文件的信息。一般情况下，NKD 编译系统会搜寻 $NDK/sources/*/目录中的所有 Android.mk 文件，但如果程序开发人员将 Android.mk 文件放置在下一级目录中，则需要在上一级目录中的 Android.mk 文件添加如下代码：

```
include $ (call all-subdir-makefiles)
```

下面来分析 AndroidNdkDemo 示例的 add-module 模块的 Android.mk 文件。

Android. mk 文件的代码如下：

```
1    LOCAL_PATH:=$ (call my-dir)
2
3    include $ (CLEAR_VARS)
4
5    LOCAL_MODULE     := add-module
6    LOCAL_SRC_FILES := add-module.c
7
8    include $ (BUILD_SHARED_LIBRARY)
```

每个 Android. mk 文件都必须以第 1 行代码开始，变量 LOCAL_PATH 用来定义需要编译的 C/C++源代码的位置，my-dir 由 NKD 编译系统提供，表示当前目录的位置。代码第 3 行的 include ＄(CLEAR_VARS)表示清空所有以 LOCAL_开始的变量，例如 LOCAL_MODULE、LOCAL_SRC_FILES、LOCAL_STATIC_LIBRARIES 等，但第 1 行定义的 LOCAL_PATH 不在清空的范围内。因为所有的脚本都将粘贴到同一个 GNU Make 的执行上下文中，而且所有变量都是全局变量，因此必须在每次使用前清空所有以前用过的变量。

第 5 行代码变量 LOCAL_MODULE 用来声明模块名称，模块名称必须唯一，而且中间不能存在空格。NKD 编译系统将会在模块名称前自动添加 lib 前缀，然后生产 so 文件。这里的模块名称为 add-module，生产的共享库文件名为 libadd-module. so。但需要注意的是，如果程序开发人员使用具有 lib 前缀的模块名称，NKD 编译系统将不再添加前缀，例如模块名称为 libsub，生产的共享库文件名为 libsub. so。

第 6 行代码中的变量 LOCAL_SRC_FILES 表示编译模块所需要使用的 C/C++文件列表，但不需要给出头文件的列表，因为 NKD 编译系统会自动计算依赖关系。add-module 模块仅需要一个 C 文件，文件名为 add-module. c。默认情况下，结尾名为. c 的文件是 C 语言源文件，结尾名为. cpp 的文件是 C++语言源文件。

第 8 行代码 include ＄(BUILD_SHARED_LIBRARY)表示 NKD 编译系统构建共享库，如果变量 BUILD_SHARED_LIBRARY 更改为 BUILD_STATIC_LIBRARY，则表示需要 NKD 编译系统构建静态库。

### 4. 建立 C 源代码文件

根据 Android. mk 文件的声明，add-module 模块仅包含一个 C 源代码文件 add-module. c。add-module. c 文件的作用是实现两个整数加法运算功能，全部代码如下：

```
#include <jni.h>

jlong Java_edu_hrbeu_AndroidNdkDemo_AndroidNdkDemo_add( JNIEnv*   env,
                                    jobject   this, jlong     x, jlong      y )
{
    return x+y;
```

```
6    }
```

代码第 1 行引入的是 JNI(Java Native Interface)的头文件。第 3 行代码是函数名称,jlong 表示 Java 长型整数,Java_edu_hrbeu_AndroidNdkDemo_AndroidNdkDemo_add 的构成为 Java_<包名称>_<类>_<函数>,其中<函数>的名称和参数要与 AndroidNdkDemo.java 文件定义的函数一致,AndroidNdkDemo.java 文件定义的函数为 public native long add(long x, long y)。第 5 行代码用来返回加法运算结果。

### 5. 编译共享库模块

到目前为止,编译前的准备工作基本就绪,程序开发人员可以直接编译共享库模块了。首先启动 cygwin,然后切换到 Androd NDK 的主目录下,输入如下的编译命令:

```
make APP=ndk-demo
```

其中,ndk-demo 是 apps 目录下的应用程序目录名称。在指定应用程序(目录)名称后,NKD 编译系统会首先找到目录中的 Application.mk 文件,根据 Application.mk 文件的信息,确定该 Android 共享需要使用 add-module 模块;然后在 sources 目录中搜索所有 Android.mk 文件,在找到与 add-module 模块匹配的 Android.mk 文件后,根据 Android.mk 文件提供的信息编译指定的 C/C++源代码文件,形成共享库文件;最后将生产的共享库文件拷贝到 Android 工程的指定目录中。

但目前 Android NDK 1.5 版本存在一个小错误,如果用户直接运行 make APP=ndk-demo 命令编译程序,NDK 编译系统将提示 out/host/config.mk 文件不存在,让用户重新运行 build/host-setup.sh 脚本,如图 10.6 所示。

```
/cygdrive/e/android/android-ndk-1.5_r1
$
build/core/main.mk:41: *** Android NDK:  Aborting    .  Stop.

warner@PC-200807080033 /cygdrive/e/android/android-ndk-1.5_r1
$ make APP=ndk-demo
Android NDK:  The configuration file 'out/host/config.mk' doesnt' exist.
Android NDK:  Please run 'build/host-setup.sh' to generate it.
build/core/main.mk:41: *** Android NDK:  Aborting    .  Stop.

warner@PC-200807080033 /cygdrive/e/android/android-ndk-1.5_r1
$
```

**图 10.6 Android NDK 1.5 版本的错误提示**

解决这一问题的方法非常简单,只需将<Android NDK>/build/out/host 文件夹和其中的 config.mk 文件拷贝到<Android NDK>/out 目录中即可。

编译成功的提示信息如图 10.7 所示,提示信息包括编译 add-module 模块所使用到的文件,生产 so 文件的文件名和 so 文件的安装位置。为了确认是否成功编译了模块,用户可以打开 apps/ndk-demo/project/libs/armeabi 目录,如果目录中存在 libadd-module.so 文件,则表示编译成功。




图 10.7 编译成的提示

### 6. 运行 Android 程序

在运行 AndroidNdkDemo 示例程序前，务必将 AndroidNdkDemo.java 文件中第 16 行和第 23 行的注释取消，并注释掉第 25 行到第 27 行代码。

## 习 题

1. 简述 Android NDK 开发的优势和不足。
2. 说明 Android NDK 应用程序开发的一般步骤。
3. 参考 NDK 的 tow-libs 示例，使用静态库实现 AndroidNdkDemo 示例中加法运算的函数功能。
4. 使用 NDK 能提高复杂函数的运算速度，但程序运行效率的提升并不容易度量。分别使用 C/C++和 Java 语言设计一个具有复杂运算的函数，通过对比函数的调用和返回时间，分析 NDK 对提高程序运行效率的能力。

# 第 11 章 chapter 11

# 综合示例设计与开发

本章将以“天气预报短信服务软件”作为示例，综合运用以往章节所学习的知识和技巧，从需求分析、界面设计、模块设计和程序开发等几个方面，详细介绍 Android 应用程序的设计思路与开发方法。通过本章的学习可以让读者初步具备 Android 应用程序的设计和开发能力。

**本章学习目标：**

- 了解 Android 应用程序的设计和开发过程
- 掌握使用多种组件进行 Android 程序开发的方法

## 11.1 需求分析

通过前面章节的学习，读者应该已经掌握了较多的 Android 应用程序的开发知识和方法，但如何能够综合地运用这些知识和方法，解决实际开发中所遇到的问题，还是一个需要继续学习和探讨的问题。设计本章的初衷就是希望读者能够根据实际项目的需求，准确地分析出 Android 应用程序开发可能涉及到的知识点，并学会如何通过分析软件的需求，快速地设计出应用程序的用户界面和模块结构，并最终完成应用程序的开发和调试。

本章提供的“天气预报短信服务软件”是一个略微复杂的示例。在这个综合示例中，有一个显示天气情况的用户界面，可以通过图片和文字显示当前和未来几天的天气状况，包括温度、湿度、风向和雨雪情况等。这些天气数据是通过后台服务获取的，这个后台服务可以按照一定时间间隔，从 Google 上获取天气预报信息，并将天气信息保存在后台服务中。示例还需要提供基于 SMS 短信的天气数据服务，其他手机用户可以向本示例所在的手机上发送 SMS 短信，并在短信中包含用户指定的关键字，则可以将保存在后台服务中的天气情况，再通过 SMS 短信回复给用户。最后，每个被发送的 SMS 短信都要被记录下来，用户可以浏览或删除这些回复信息。

从上面的描述中可以基本了解软件的功能需求，但为了将需求分析变得简单明了，首先找出用户界面上需要显示的内容。上面的功能描述中有显示天气情况的界面和浏览回复信息的界面，除此以外，一般情况下应用软件还包含设置配置信息的用户界面。因此，本示例应该包含三个主要的用户界面：

(1) 显示天气预报的用户界面。

(2) 显示已发送 SMS 短信的用户界面。

(3) 浏览和设置配置信息的用户界面。

进一步从用户界面出发,分析隐藏在用户界面后面的内部功能,这些功能是用户界面能够正确实现的基础。在"显示天气预报的用户界面"中,如果需要在界面上显示天气信息,则必须首先从互联网获取天气信息数据,并将数据提供给自动回复的 SMS 短信。而在"显示已发送 SMS 短信的用户界面"中,为了实现自动回复具有关键字的 SMS 短信,则需要根据关键字监视所有接收到的短信,并且支持发送 SMS 短信功能,最后为能够浏览回复的短信信息,还需要将 SMS 数据保存到数据库中。在"浏览和设置配置信息的用户界面"中,则应该实现基本的配置信息保存和读取功能,并能够恢复系统的默认设置。根据用户的功能需求,用户界面和内部功能的关系如下所示:

(1) 显示天气预报的用户界面

获取 Google 的天气数据。

(2) 显示 SMS 短信的用户界面

① 根据关键字监视 SMS 短信。

② 发送包含天气信息的 SMS 短信。

② 将发送的 SMS 短信写入数据库。

(3) 浏览和设置配置信息的用户界面

① 将用户设置的配置信息保存到数据库。

② 启动时读取数据库中的配置信息。

③ 恢复默认设置。

在程序的需求分析完成后,读者已经基本掌握了"天气预报短信服务软件"在用户界面和内部功能方面的需求。下一步工作是根据用户界面的需求,详细设计每一个用户界面的具体内容,以及软件的功能模块划分和模块之间的调用关系。

## 11.2 程序设计

### 11.2.1 用户界面设计

根据需求中的用户界面分析,应用程序应包含三个主要的用户界面,这里需要进一步分析每个用户界面中应该包含哪些显示内容。

在"显示天气预报的用户界面"中,根据 Google 可以提供的数据,在界面上可以显示当前的天气状况,包括城市名称、温度、湿度、风向、雨雪情况和获取数据时间等信息;还可以显示未来 4 天的天气状况,但仅包括温度和雨雪情况。在"显示已发送 SMS 短信的用户界面"中,应显示每个回复短信的时间、目标手机号码、城市名称、当天的天气状况和未来一天的天气状况。在"浏览和设置配置信息的用户界面"中,应显示获取天气预报的目标城市名称、获取数据的频率和短信的关键字,并允许用户设置是否提供短信服务以及是否记录回复的短信信息。

根据对用户界面显示内容的分析，绘制出用户界面的草图，如图 11.1 所示。

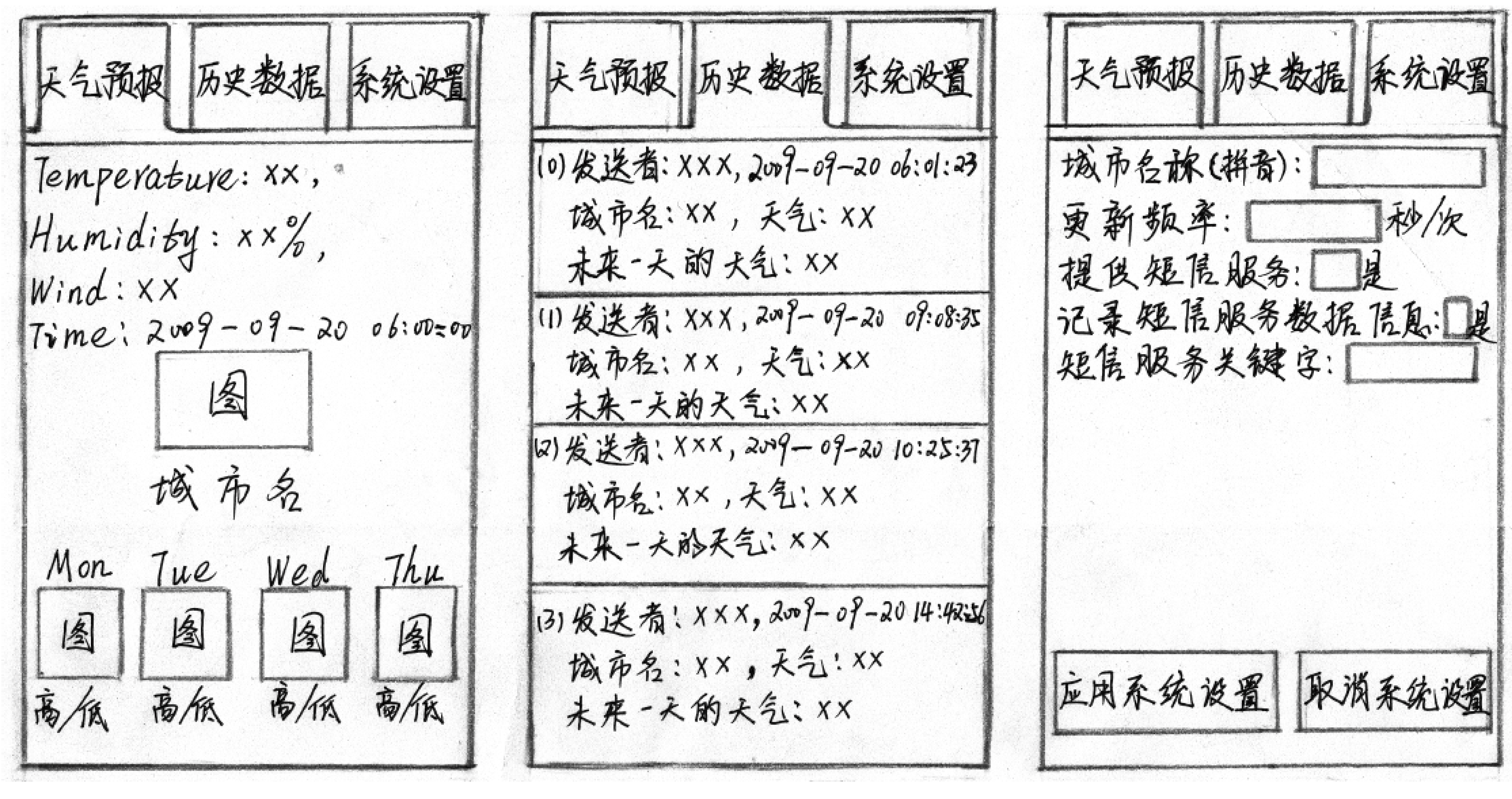



**图 11.1 用户界面的草图**

在初步完成的用户界面设计后，下一步进入应用程序的数据库设计。

## 11.22 数据库设计

本示例主要有两种数据需要存储，一个是配置信息，另一个是 SMS 短信服务信息。因为配置信息的数据量很小，从 Android 支持的存储方式上分析，可以保存在 SharePreference、文件或 SQLite 数据库中。SMS 短信服务信息是一个随着时间推移而不断增加的数据，属于文本信息，而且有固定的格式，因此适合使用 SQLite 数据库进行存储。综合分析这两种需要存储的数据，选择 SQLite 数据库作为存储数据的方法。

配置信息中主要保存天气信息查询的城市名称，访问 Google 更新天气信息的频率，请求天气信息 SMS 短信的关键字，以及是否提供短信服务和是否记录短信服务内容。配置信息的数据库表结构如表 11.1 所示。

**表 11.1 配置信息的数据库表结构**

| 属　　性 | 数据类型 | 说　　明 |
|---|---|---|
| _id | integer | 自动增加的主键 |
| city_name | text | 进行天气信息查询的城市名 |
| refresh_speed | text | 进行天气信息查询的频率，单位为秒/次 |
| sms_service | text | 是否提供短信服务，即接收到请求短信后是否回复包含天气信息的短信 |
| sms_info | text | 是否记录发出的 SMS 短信的信息 |
| key_word | text | 短信服务的关键字，用以确定哪条短信是请求天气服务的短信 |

SMS 短信服务信息主要保存请求服务短信的发送者、短信内容、接收时间和回复信息的内容。SMS 短信服务信息的数据库表结构如表 11.2 所示。

**表 11.2　SMS 短信服务信息的数据库表结构**

| 属　　性 | 数 据 类 型 | 说　　明 |
|---|---|---|
| _id | integer | 自动增加的主键 |
| sms_sender | text | 请求服务短信的发送者 |
| sms_body | text | 请求服务短信的内容信息 |
| sms_receive_time | text | 接收到请求服务短信的时间 |
| return_result | text | 回复短信的内容 |

## 11.2.3　程序模块设计

从功能需求上分析，可以将整个应用程序划分为 4 个模块，分别是用户界面、后台服务、数据库适配器和短信监听器，各模块之间的关系如图 11.2 所示。

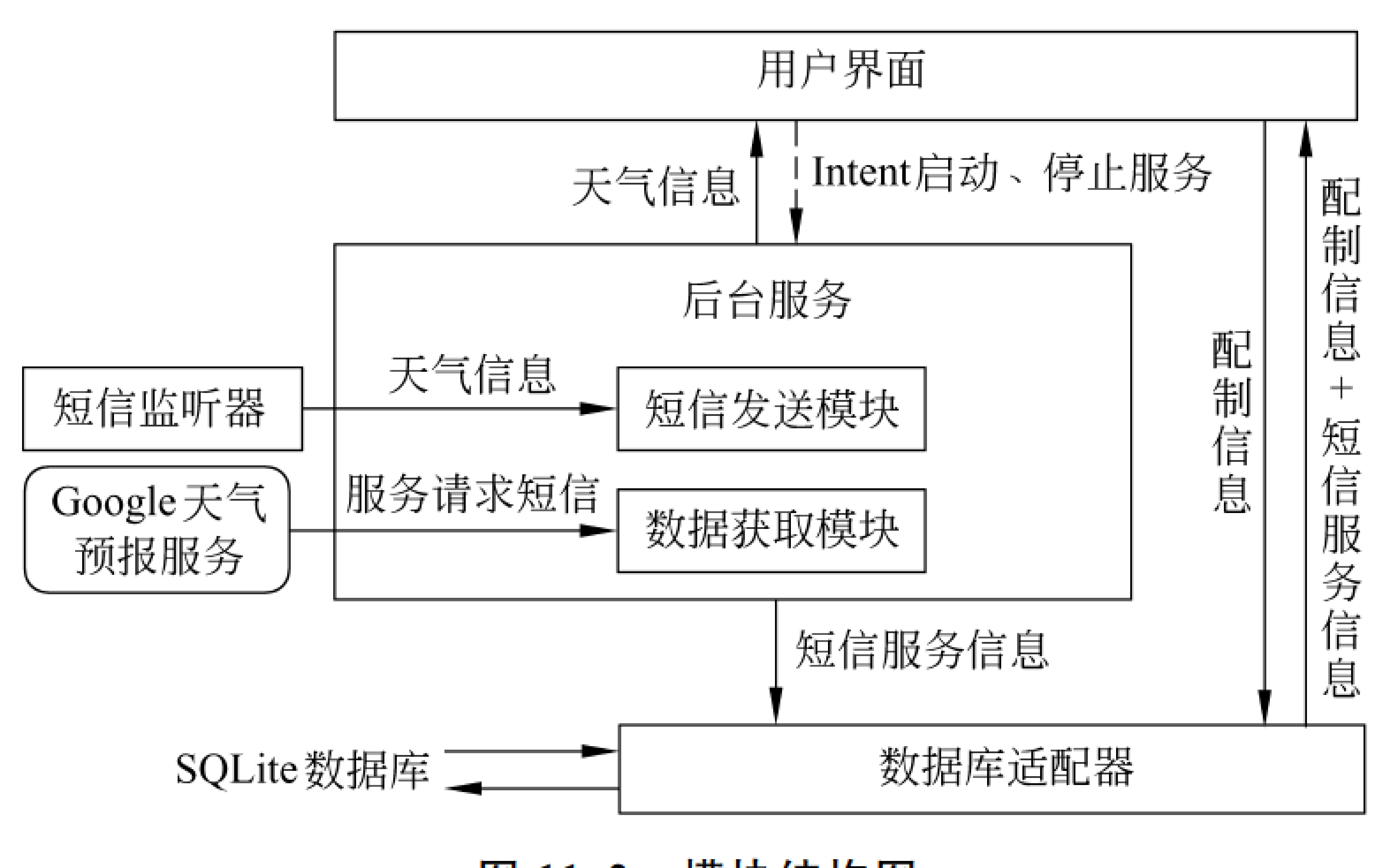



**图 11.2　模块结构图**

从模块结构图中不难看出，后台服务是整个应用程序的核心，主要包含两个子模块，一个是“数据获取模块”，负责周期性的从 Google 获取天气信息；另一个是“短信服务模块”，负责处理接收到的服务请求短信，并发送包含天气信息的短信。后台服务由用户界面通过 Intent 启动，启动后的后台服务可以在用户界面关闭后仍然保持运行状态，直到用户通过用户界面发送 Intent 停止服务，或系统因资源不足而强行关闭服务。

用户界面从后台服务获取天气信息，而没有直接通过网络访问 Google 的天气数据。之所以这么设计，一方面是因为后台服务使用了工作线程，通过后台服务获取天气数据可以避免因网络通信不畅造成界面失去响应；另一方面，在用户关闭界面后，后台服务仍然需要更新天气信息，以保证短信服务数据的准确性。用户界面还会调用数据库适配器，向 SQLite 数据库中写入、读取配置信息，或对 SMS 短信服务信息进行操作。

短信监听器是一个 BroadcastReceiver，监视所有接收到的短信。如果短信中包含用户

自定义的关键字，短信监听器则会认为这条短信是天气服务请求短信，将短信的相关信息写入后台服务的短信服务队列。当然，如果用户在配置信息中选择无需提供短信服务，短信监听器仍然继续监听所有短信，只是后台服务不再允许将服务请求短信写入服务队列。

数据库适配器封装了所有对 SQLite 数据库操作的方法，用户界面和后台服务会调用它实现数据库操作。

在完成用户界面设计、数据库设计和模块设计后，程序设计阶段基本完成，下面进入程序开发阶段。

## 11.3 程序开发

### 11.3.1 文件结构与用途

在程序开发阶段，首先确定“天气预报短信服务软件”的工程名称为 WeatherDemo，包名称为 edu. hrbeu. WeatherDemo，据程序模块设计的内容，建立 WeatherDemo 示例。WeatherDemo 示例源代码的文件结构如图 11.3 所示。

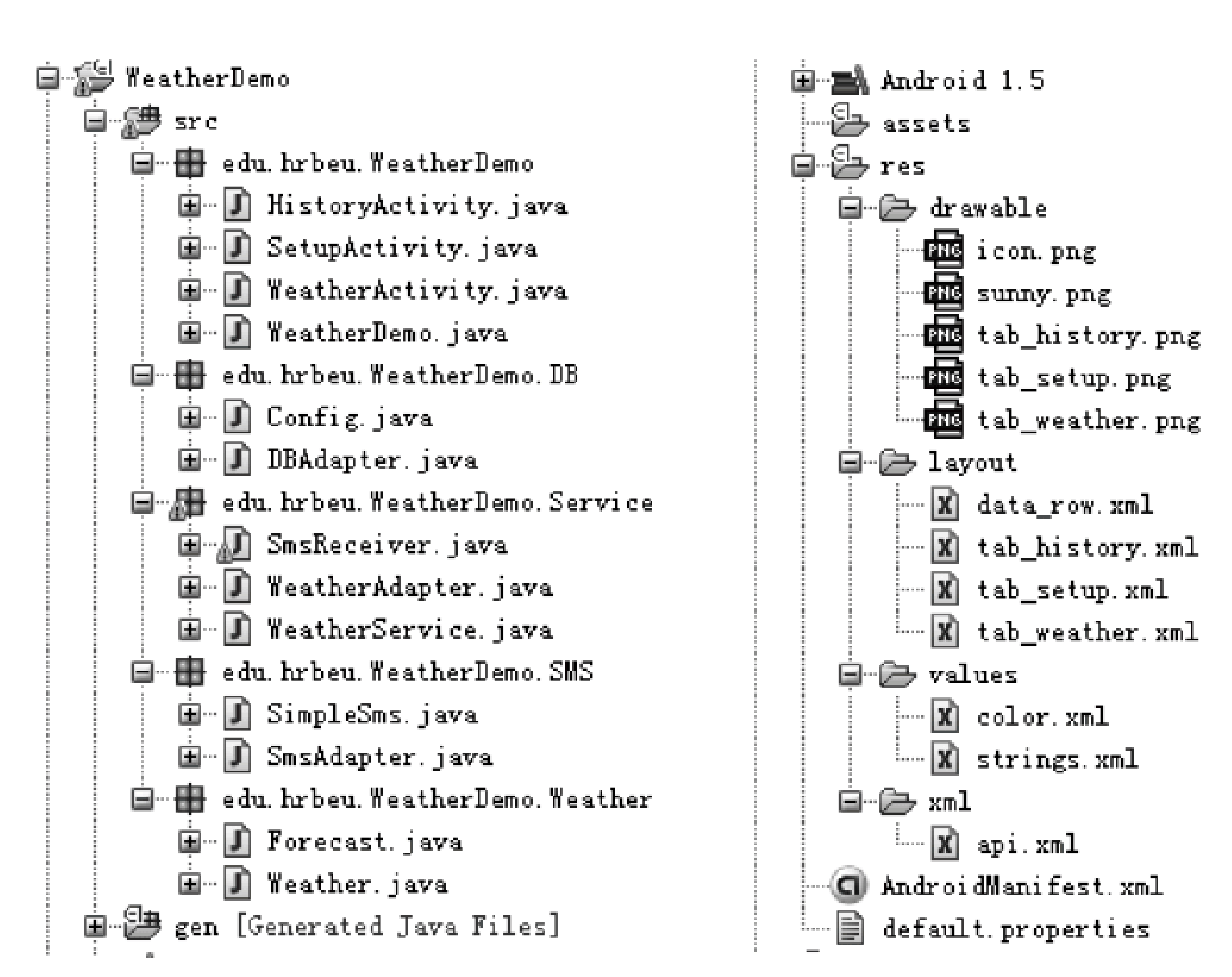



**图 11.3　WeatherDemo 示例的源代码文件**

为了使源代码文件的结构更加清晰，WeatherDemo 示例设置了多个命名空间，分别用来保存用户界面、数据库、后台服务、SMS 短信和天气数据的源代码文件，命名空间的名称以及说明参考表 11.3。

**表 11.3　WeatherDemo 示例的命名空间**

| 命名空间 | 说明 |
| --- | --- |
| edu. hrbeu. WeatherDemo | 存放与用户界面相关的源代码文件 |
| edu. hrbeu. WeatherDemo. DB | 存放与 SQLite 数据库相关的源代码文件 |
| edu. hrbeu. WeatherDemo. Service | 存放与后台服务相关的源代码文件 |

续表

| 命 名 空 间 | 说 明 |
|---|---|
| edu. hrbeu. WeatherDemo. SMS | 存放与 SMS 短信相关的源代码文件 |
| edu. hrbeu. WeatherDemo. Weather | 存放与天气数据有关的源代码文件 |

WeatherDemo 示例将不同用途的源代码文件放置在不同的命名空间中,源代码文件的名称和用途可以参考表 11.4。

**表 11.4 WeatherDemo 示例的文件用途说明**

| 包 名 称 | 文 件 名 | 说 明 |
|---|---|---|
| . WeatherDemo | HistoryActivity. java | “历史数据”页的 Activity |
| | SetupActivity. java | “系统设置”页的 Activity |
| | WeatherActivity. java | “天气预报”页的 Activity |
| | WeatherDemo. java | 程序启动默认的 Activity |
| . WeatherDemo. DB | Config. java | 保存配置信息的类 |
| | DBAdapter. java | 数据库适配器 |
| . WeatherDemo. Service | SmsReceiver. java | 短信监听器 |
| | WeatherAdapter. java | 数据获取模块 |
| | WeatherService. java | 后台服务 |
| . WeatherDemo. SMS | SimpleSms. java | 简化的 SMS 短信类 |
| | SmsAdapter. java | 短信发送模块 |
| . WeatherDemo. Weather | Forecast. java | 未来天气信息的类 |
| | Weather. java | 当前天气信息的类 |

Android 的资源文件保存在/res 的子目录中。其中/res/drawable 目录中保存的是图像文件,/res/layout 目录中保存的是布局文件,/res/values 目录中保存的是用来定义字符串和颜色的文件,/res/xml 目录保存的是 XML 格式的数据文件。所有在程序开发阶段可以被调用的资源都保存在这些目录中,具体每个资源文件的用途可以参考表 11.5。

**表 11.5 资源文件名称与用途**

| 资源目录 | 文 件 | 说 明 |
|---|---|---|
| drawable | icon. png | 图标文件 |
| | sunny. png | 调试用的天气图片 |
| | tab_history. png | TabHost 中“历史数据”页的图片 |
| | tab_setup. png | TabHost 中“系统设置”页的图片 |
| | tab_weather. png | TabHost 中“天气预报”页的图片 |

续表

| 资源目录 | 文　　件 | 说　　明 |
|---|---|---|
| layout | data_row. xml | “历史数据”页 ListActivity 的每行数据的布局 |
| | tab_history. xml | TabHost 中“历史数据”页的布局 |
| | tab_setup. xml | TabHost 中“系统设置”页的布局 |
| | tab_weather. xml | TabHost 中“天气预报”页的布局 |
| values | color. xml | 保存颜色的 XML 文件 |
| | string. xml | 保存字符串的 XML 文件 |
| xml | api. xml | 从 Google 下载的天气数据文件。在程序运行时没有实际作用，但在开发过程中可以让读者了解数据格式 |

在定义了所有文件和类的用途后，下一步将依据程序模块结构图（见图 11.2），按照自底向上的顺序对每个模块进行详细的介绍。自底向上介绍有利于理解模块之间的调用关系，也避免了在介绍上层模块时，读者不了解所调用的下层模块。

## 11.3.2 数据库适配器

数据库适配器是最底层的模块，主要用于封装用户界面和后台服务对 SQLite 数据库的操作。数据库适配器的核心代码主要在 DBAdapter. java 文件中，在介绍数据库适配器的核心代码前，首先了解一下用户保存配置信息的类文件 Config. java。

Config. java 文件的全部代码如下：

```
package edu.hrbeu.WeatherDemo.DB;

public class Config {
	public static String CityName;
	public static String RefreshSpeed;
	public static String ProvideSmsService;
	public static String SaveSmsInfo;
	public static String KeyWord;

	public static void LoadDefaultConfig(){
		CityName= "New York";
		RefreshSpeed= "60";
		ProvideSmsService= "true";
		SaveSmsInfo= "true";
		KeyWord= "NY";
	}
}
```

从代码中不难看出，公有静态属性 CityName、RefreshSpeed、ProvideSmsService、SaveSmsInfo 和 KeyWord，完全对应数据库中保存配置信息表的属性（见表 11.1）。在程

序启动后，保存在数据库中的城市名称、更新频率、是否提供短信服务、是否保存短信信息和关键字等内容，将被写入到这个 Config 类中，供其他模块在作逻辑判断时使用。

代码第 10 行的 LoadDefaultConfig()函数，保存了程序内置的配置参数。此函数会在两个情况下被调用，一是用户主动选择“恢复默认设置”；二是首次启动程序时，用来初始化保存配置参数的数据库。

DBAdapter 类与以往介绍过的数据库适配器类相似，都具有继承 SQLiteOpenHelper 的帮助类 DBOpenHelper。DBOpenHelper 在建立数据库时，同时建立两个数据库表，并对保存配置信息的表进行了初始化，初始化的相关代码在第 42～49 行。

```
1	private static final String DB_NAME= "weather_app.db";
2	private static final String DB_TABLE_CONFIG= "setup_config";
3	private static final String DB_CONFIG_ID= "1";
4	private static final int DB_VERSION= 1;
5	
6	public static final String KEY_ID= "_id";
7	public static final String KEY_CITY_NAME= "city_name";
8	public static final String KEY_REFRESH_SPEED= "refresh_speed";
9	public static final String KEY_SMS_SERVICE= "sms_service";
10	public static final String KEY_SMS_INFO= "sms_info";
11	public static final String KEY_KEY_WORD= "key_word";
12	
13	private static final String DB_TABLE_SMS= "sms_data";
14	public static final String KEY_SENDER= "sms_sender";
15	public static final String KEY_BODY= "sms_body";
16	public static final String KEY_RECEIVE_TIME= "sms_receive_time";
17	public static final String KEY_RETURN_RESULT= "return_result";
18	
19	/** 静态 Helper 类，用于建立、更新和打开数据库 */
20	private static class DBOpenHelper extends SQLiteOpenHelper {
21	public DBOpenHelper(Context context, String name, CursorFactory factory, int version) {
22	        super(context, name, factory, version);
23	        }
24	
25	    private static final String DB_CREATE_CONFIG= "create table "+
26	        DB_TABLE_CONFIG+ " ("+ KEY_ID+ " integer primary key autoincrement, "+
27	        KEY_CITY_NAME+ " text not null, "+ KEY_REFRESH_SPEED+ " text,"+
28	        KEY_SMS_SERVICE+ " text, "+ KEY_SMS_INFO+ " text, "+
29	        KEY_KEY_WORD+ " text);";
30	
31	    private static final String DB_CREATE_SMS= "create table "+
32	        DB_TABLE_SMS+ " ("+ KEY_ID+ " integer primary key autoincrement, "+
```

```
            KEY_SENDER+" text not null, "+KEY_BODY+" text, "+
            KEY_RECEIVE_TIME+" text, "+KEY_RETURN_RESULT+" text);";

        @Override
        public void onCreate(SQLiteDatabase _db) {
            _db.execSQL(DB_CREATE_CONFIG);
            _db.execSQL(DB_CREATE_SMS);

            //初始化系统配置的数据表
            Config.LoadDefaultConfig();
            ContentValues newValues=new ContentValues();
            newValues.put(KEY_CITY_NAME, Config.CityName);
            newValues.put(KEY_REFRESH_SPEED, Config.RefreshSpeed);
            newValues.put(KEY_SMS_SERVICE, Config.ProvideSmsService);
            newValues.put(KEY_SMS_INFO, Config.SaveSmsInfo);
            newValues.put(KEY_KEY_WORD, Config.KeyWord);
            _db.insert(DB_TABLE_CONFIG, null, newValues);
        }

        @Override
        public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion) {
            _db.execSQL("DROP TABLE IF EXISTS "+DB_TABLE_CONFIG);
            _db.execSQL("DROP TABLE IF EXISTS "+DB_CREATE_SMS);
            onCreate(_db);
        }
    }
```

在DBAdapter类中，用户界面会调用SaveConfig()和LoadConfig()，向SQLite数据库中保存和读取配置信息。保存配置信息时，SaveConfig()函数会将Config类中的公有静态属性写入数据库；反之，LoadConfig()会将数据库中的配置信息写入Config类中的公有静态属性。SaveConfig()和LoadConfig()的代码如下：

```
    public void SaveConfig(){
        ContentValues updateValues=new ContentValues();
        updateValues.put(KEY_CITY_NAME, Config.CityName);
        updateValues.put(KEY_REFRESH_SPEED, Config.RefreshSpeed);
        updateValues.put(KEY_SMS_SERVICE, Config.ProvideSmsService);
        updateValues.put(KEY_SMS_INFO, Config.SaveSmsInfo);
        updateValues.put(KEY_KEY_WORD, Config.KeyWord);
        db.update(DB_TABLE_CONFIG, updateValues,  KEY_ID+"="+DB_CONFIG_ID, null);
        Toast.makeText(context, "系统设置保存成功", Toast.LENGTH_SHORT).show();
    }

    public void LoadConfig() {
```

```
        Cursor result=db.query(DB_TABLE_CONFIG, new String[] { KEY_ID, KEY_CITY_NAME,
            KEY_REFRESH_SPEED,KEY_SMS_SERVICE, KEY_SMS_INFO, KEY_KEY_WORD},
            KEY_ID+ "="+ DB_CONFIG_ID, null, null, null, null);
        if (result.getCount()==0 || !result.moveToFirst()){
                return;
            }
        Config.CityName=result.getString(result.getColumnIndex(KEY_CITY_NAME));
        Config.RefreshSpeed=result.getString(result.getColumnIndex(KEY_REFRESH_SPEED));
        Config.ProvideSmsService=result.getString(result.getColumnIndex(KEY_SMS_SERVICE));
        Config.SaveSmsInfo=result.getString(result.getColumnIndex(KEY_SMS_INFO));
        Config.KeyWord=result.getString(result.getColumnIndex(KEY_KEY_WORD));
        Toast.makeText(context, "系统设置读取成功", Toast.LENGTH_SHORT).show();
    }
```

另一个会调用DBAdapter类的是后台服务，即WeatherService类。后台服务主要调用SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数，分别用来保存SMS短信记录、删除所有SMS数据记录和获取所有SMS数据记录。在GetAllSms()函数中，调用了一个私有函数ToSimpleSms(Cursor cursor)，用来将从数据库获取的数据转换为SimpleSms对象数组。SimpleSms类将在11.3.3节进行介绍，下面先给出SaveOneSms(SimpleSms sms)、DeleteAllSms()和GetAllSms()函数的代码：

```
    public void SaveOneSms(SimpleSms sms){
        ContentValues newValues=new ContentValues();
        newValues.put(KEY_SENDER, sms.Sender);
        newValues.put(KEY_BODY, sms.Body);
        newValues.put(KEY_RECEIVE_TIME, sms.ReceiveTime);
        newValues.put(KEY_RETURN_RESULT, sms.ReturnResult);
        db.insert(DB_TABLE_SMS, null, newValues);
    }
    public long DeleteAllSms() {
        return db.delete(DB_TABLE_SMS, null, null);
    }
    public SimpleSms[] GetAllSms() {
        Cursor results=db.query(DB_TABLE_SMS, new String[] { KEY_ID, KEY_SENDER,
            KEY_BODY, KEY_RECEIVE_TIME, KEY_RETURN_RESULT},
            null, null, null, null, null);
         return ToSimpleSms(results);
    }
    private SimpleSms[] ToSimpleSms(Cursor cursor){
        int resultCounts=cursor.getCount();
        if (resultCounts==0 || !cursor.moveToFirst()){
            return null;
        }

```

```
        SimpleSms[] sms= new SimpleSms[resultCounts];
        for (int i= 0 ; i< resultCounts; i++){
            sms[i]= new SimpleSms();
            sms[i].Sender= cursor.getString(cursor.getColumnIndex(KEY_SENDER));
            sms[i].Body= cursor.getString(cursor.getColumnIndex(KEY_BODY));
            sms[i].ReceiveTime= cursor.getString(cursor.getColumnIndex(KEY_RECEIVE_TIME));
        sms[i].ReturnResult= cursor.getString(cursor.getColumnIndex(KEY_RETURN_RESULT));
            cursor.moveToNext();
        }
        return sms;
    }
```

### 11.3.3 短信监听器

短信监听器本质上是 BroadcastReceiver，用于监听 Android 系统所接收到的所有 SMS 短消息，可以在应用程序关闭后仍然继续运行，核心代码在 SmsReceiver.java 文件中。同样在介绍 SmsReceiver 类前，先说明用来保存 SMS 短信内容和相关信息的 SimpleSms 类。android.telephony.gsm.SmsMessage 是 Android 提供的短信类，但这里需要一个更精简、小巧的类，保存少量的信息，因此构造了 SimpleSms 类，仅用来保存短信的发送者、内容、接收时间和返回结果。这里的“返回结果”指的是返回包含天气信息的短信内容。

SimpleSms.java 文件完整代码如下：

```
1    package edu.hrbeu.WeatherDemo.SMS;
2    import java.text.SimpleDateFormat;
3
4    public class SimpleSms {
5        public String Sender;
6        public String Body;
7        public String ReceiveTime;
8        public String ReturnResult;
9
10       public SimpleSms(){
11       }
12       public SimpleSms(String sender, String body){
13           this.Sender= sender;
14           this.Body= body;
15           SimpleDateFormat tempDate= new SimpleDateFormat("yyyy-MM-dd"+" "+"hh:mm:ss");
16           this.ReceiveTime= tempDate.format(new java.util.Date());
17           this.ReturnResult= "";
18       }
19   }
```

代码第 5 行到第 8 行的属性 Sender、Body、ReceiveTime 和 ReturnResult，分别表示

SMS 短信的发送者、内容、接收时间和返回结果。代码的第 15 行和第 16 行在 SimpleSms 类的构造函数中，直接将系统时间以“年-月-日 小时:分:秒”的格式保存在 ReceiveTime 属性中。

SmsReceiver 类继承 BroadcastReceiver，重载了 onReceive()函数。系统消息的识别和关键字的识别并不复杂，只要接收 android. provider. Telephony. SMS_RECEIVED 类型的系统消息，则表明是 Android 系统接收到了短信；将短信的内容拆分后，判断消息内容是否是配置信息中定义的关键字，即可判断该短信是否为天气服务请求短信。下面给出 SmsReceiver. java 文件的核心代码：

```
1   public class SmsReceiver extends BroadcastReceiver{
2       private static final String SMS_ACTION= "android.provider.Telephony.SMS_RECEIVED";
3
4       @ Override
5       public void onReceive(Context context, Intent intent){
6           if (intent.getAction().equals(SMS_ACTION)){
7               Bundle bundle= intent.getExtras();
8               if (bundle != null){
9                   Object[] objs= (Object[]) bundle.get("pdus");
10                  SmsMessage[] messages= new SmsMessage[objs.length];
11                  for (int i= 0; i< objs.length; i++){
12                      messages[i]= SmsMessage.createFromPdu((byte[]) objs[i]);
13                  }
14                  String smsBody= messages[0].getDisplayMessageBody();
15                  String smsSender= messages[0].getDisplayOriginatingAddress();
16                  if (smsBody.trim().equals(Config.KeyWord) && Config.ProvideSmsService.equals
                    ("true")){
17                      SimpleSms simpleSms= new SimpleSms(smsSender, smsBody);
18                      WeatherService.RequerSMSService(simpleSms);
19                      Toast.makeText(context, "接收到服务请求短信", Toast.LENGTH_SHORT).
                        show();
20                  }
21              }
22          }
23      }
24  }
```

代码第 10 行将带有 pdus 字符串特征的对象，通过 Bundle. get()函数提取出来，并在代码第 12 行使用 SmsMessage. CreateFromPdu()函数构造 SmsMessage 对象。在代码第 11 行使用循环语句是因为接收到的短信可能不止一条，但从第 14 行和第 15 行代码看，这里只处理第 1 条短信。代码第 17 行构造 SimpleSms 对象，然后在代码第 18 行调用 WeatherService 类的 RequerSMSService()函数，将 SimpleSms 对象添加到短信队列中。

最后，在 AndroidManifest. xml 文件中注册短信监听器 SmsReceiver，并声明可以接

收短信的用户许可 android. permission. RECEIVE_SMS。需要注意的是如果注册的组件不在根命名空间中，则需要将子命名空间写在类的前面，例如下面在代码第 1 行中，因为 SmsReceiver. java 文件在 edu. hrbeu. WeatherDemo. Service 命名空间下，而不在根命名空间 edu. hrbeu. WeatherDemo 下，因此注册组件时需要在类名 SmsReceiver 前添加. Service。

```
1   <receiver android:name=".Service.SmsReceiver" >
2       <intent-filter>
3           <action android:name="android.provider.Telephony.SMS_RECEIVED" />
4       </intent-filter>
5   </receiver>
6   <uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

## 11.3.4 后台服务

后台服务是 WeatherDemo 示例的核心模块，在用户启动后持续在后台运行，直到用户手动停止服务。后台服务主要有两个功能，一是发送包含天气信息的 SMS 短信（短信发送模块），二是周期性的获取 Google 的天气数据（数据获取模块）。

### 1. 短信发送模块

后台服务在单独的线程上运行，首先调用 ProcessSmsList()函数，检查短信队列中是否有需要回复的短信，然后调用 GetGoogleWeatherData()函数获取天气数据，最后线程暂停 1 秒，以释放 CPU 资源。WeatherDemo 示例后台服务的核心代码在 WeatherService. java 文件中，下面是线程调用函数的部分代码：

```
private static ArrayList<SimpleSms> smsList=new ArrayList<SimpleSms>();

private Runnable backgroudWork=new Runnable(){
    @Override
    public void run() {
        try{
            while(!Thread.interrupted()){
                ProcessSmsList();
                GetGoogleWeatherData();
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
};
```

ProcessSmsList()函数用来检查短信列表 smsList，并根据 Weather 类中保存的天气

数据，向请求者的发送回复短信。WeatherService.java 文件的 ProcessSmsList()函数代码如下：

```
1   private void ProcessSmsList(){
2       if (smsList.size()==0){
3           return;
4       }
5       SmsManager smsManager=SmsManager.getDefault();
6       PendingIntent mPi=PendingIntent.getBroadcast(this, 0, new Intent(), 0);
7       while(smsList.size()>0){
8           SimpleSms sms=smsList.get(0);
9           smsList.remove(0);
10          smsManager.sendTextMessage(sms.Sender, null, Weather.GetSmsMsg(), mPi, null);
11          sms.ReturnResult=Weather.GetSmsMsg();
12          SaveSmsData(sms);
13      }
14  }
```

发送短信是使用 SmsManager 对象的 sendTextMessage()方法，该方法一共需要 5 个参数，第 1 个参数是收件人地址；第 2 个参数是发件人地址；第 3 个参数是短信正文；第 4 个参数是发送服务；第 5 个参数是送达服务。sendTextMessage()方法的收件人地址和短信正文是不可为空的参数，而且一般 GSM 规范要求短信内容要控制在 70 个汉字以内。代码第 8 行的 Weather.GetSmsMsg()，用来获得供回复短信使用的天气信息，因为考虑到短信的字数限制，仅返回当天和未来一天的天气状况。下面分别给出 Weather.java 和 Forecast.java 文件的完整代码。

Weather.java 文件的代码如下：

```
package edu.hrbeu.WeatherDemo.Weather;
import android.graphics.Bitmap;

public class Weather {
    public static String city;
    public static String forecase_date;
    public static String current_date_time;
    public static String current_condition;
    public static String current_temp;
    public static String current_humidity;
    public static String current_image_url;
    public static Bitmap current_image;
    public static String current_wind;

    public static Forecast[] day=new Forecast[4];

    static {
```

```
            for (int i=0; i<day.length; i++){
                day[i]=new Forecast();
            }
        }

        public static String GetSmsMsg(){
            String msg="";
            msg+=city+",";
            msg+=current_condition+", "+current_temp+". ";
            msg+=day[0].day_of_week+", "+day[0].condition+", "+
                    day[0].high+"/"+day[0].low;
            return msg;
        }
    }
```

Forecast.java 文件的代码如下：

```
package edu.hrbeu.WeatherDemo.Weather;
import android.graphics.Bitmap;

public class Forecast {
    public String day_of_week;
    public String low;
    public String high;
    public String image_url;
    public Bitmap image;
    public String condition;
}
```

## 2. 数据获取模块

天气数据是从 Google 提供的 Web Service 中获取的，数据的获取地址是 http://www.google.com/ig/api? hl=en&weather=New%20York。其中，New%20York 表示获取纽约（New York）的天气数据，%20 表示一个空格。读者可以替换 New%20York，并将新的地址输入 Web 浏览器，在浏览器中可以直接看到 XML 格式的天气数据。

在资源目录中的/res/xml/api.xml 文件，就是 2009 年 9 月 22 日获取的纽约天气数据。在程序资源中保留 api.xml 文件，主要是用来帮助读者分析 XML 数据格式，在程序运行期间并不访问该文件。api.xml 文件的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<xml_api_reply version="1">
    <weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0" section="0"
    >
```

```
    <forecast_information>
        <city data="New York, NY"/>
        <postal_code data="New York"/>
        <latitude_e6 data=""/>
        <longitude_e6 data=""/>
        <forecast_date data="2009-09-22"/>
        <current_date_time data="2009-09-22 16:51:00+0000"/>
        <unit_system data="US"/>
    </forecast_information>

    <current_conditions>
        <condition data="Mostly Cloudy"/>
        <temp_f data="72"/>
        <temp_c data="22"/>
        <humidity data="Humidity: 71% "/>
        <icon data="/ig/images/weather/mostly_cloudy.gif"/>
        <wind_condition data="Wind: N at 6 mph"/>
    </current_conditions>

    <forecast_conditions>
        <day_of_week data="Tue"/>
        <low data="65"/>
        <high data="76"/>
        <icon data="/ig/images/weather/mostly_sunny.gif"/>
        <condition data="Partly Sunny"/>
    </forecast_conditions>

    <forecast_conditions>
        <day_of_week data="Wed"/>
        <low data="68"/>
        <high data="79"/>
        <icon data="/ig/images/weather/chance_of_storm.gif"/>
        <condition data="Chance of Storm"/>
    </forecast_conditions>

    <forecast_conditions>
        <day_of_week data="Thu"/>
        <low data="61"/>
        <high data="83"/>
        <icon data="/ig/images/weather/chance_of_storm.gif"/>
        <condition data="Chance of Storm"/>
    </forecast_conditions>

    <forecast_conditions>
```

```
48                    <day_of_week data="Fri"/>
49                    <low data="54"/>
50                    <high data="72"/>
51                    <icon data="/ig/images/weather/sunny.gif"/>
52                    <condition data="Clear"/>
53                </forecast_conditions>
54            </weather>
55        </xml_api_reply>
```

<forecast_information>标签内的数据是天气预报的城市和时间等基本信息，<current_conditions>标签内的是当时的天气状况，4 个<forecast_conditions>标签是未来 4 天的天气情况。在 api. xml 文件中，还提供了能够反映天气情况的图标地址，例如第 19 行、第 27 行和第 35 行等。

WeatherAdapter 类实现了利用 URL 获取位图的私有函数 GetURLBitmap()，以及用来下载和解析 XML 数据的公有函数 GetWeatherData()。后台服务在调用 GetWeatherData()函数解析 Google 提供的天气数据时，会不断调用 GetURLBitmap()函数，将 XML 数据中的天气图标根据图标地址下载到本地保存。GetURLBitmap()函数的代码如下：

```
1    private static Bitmap GetURLBitmap(String urlString){
2        URL url=null;
3        Bitmap bitmap=null;
4        try {
5            url=new URL("http://www.google.com"+urlString);
6        }
7        catch (MalformedURLException e){
8            e.printStackTrace();
9        }
10
11       try{
12           HttpURLConnection conn=(HttpURLConnection) url.openConnection();
13           conn.connect();
14           InputStream is=conn.getInputStream();
15           bitmap=BitmapFactory.decodeStream(is);
16           is.close();
17       }catch (IOException e){
18           e.printStackTrace();
19       }
20       return bitmap;
21   }
```

第 12 行代码构造了支持 HTTP 功能的 URLConnection，连接后在第 14 行返回字节流，第 15 行使用字节流产生位图，最后在第 16 行关闭字节流。

GetWeatherData()函数首先根据指定的 URL 地址，从网络获取字节流数据，然后调

用轻量级 XML 解析器 XmlPullParser 对天气数据进行解析，并将解析结果保存在 Weather 类的公有静态属性中。GetWeatherData()函数的代码如下：

```
public static void GetWeatherData() throws IOException, Throwable {
    String queryString="http://www.google.com/ig/api?weather="+Config.CityName;
    URL aURL=new URL(queryString.replace(" ", "%20"));
    URLConnection conn=aURL.openConnection();
    conn.connect();
    InputStream is=conn.getInputStream();

    XmlPullParserFactory factory=XmlPullParserFactory.newInstance();
    factory.setNamespaceAware(true);
    XmlPullParser parser=factory.newPullParser();
    parser.setInput(is,"UTF-8");

    int dayCounter=0;
    while(parser.next()!=XmlPullParser.END_DOCUMENT){
        String element=parser.getName();
        if (element !=null && element.equals("forecast_information")){
            while(true){
                int eventCode=parser.next();
                element=parser.getName();
                if (eventCode==XmlPullParser.START_TAG){
                    if (element.equals("city")){
                        Weather.city=parser.getAttributeValue(0);
                    }else if (element.equals("current_date_time")){
                        Weather.current_date_time=parser.getAttributeValue(0);
                    }
                }

                if (element.equals("forecast_information") &&
                        eventCode==XmlPullParser.END_TAG){
                    break;
                }
            }
        }
        if (element !=null && element.equals("current_conditions")){
            while(true){
                int eventCode=parser.next();
                element=parser.getName();
                if (eventCode==XmlPullParser.START_TAG){
                    if (element.equals("condition")){
                        Weather.current_condition=parser.getAttributeValue(0);
                    }else if (element.equals("temp_f")){
```

```
                        Weather.current_temp=parser.getAttributeValue(0);
                    }else if (element.equals("humidity")){
                        Weather.current_humidity=parser.getAttributeValue(0);
                    }else if (element.equals("wind_condition")){
                        Weather.current_wind=parser.getAttributeValue(0);
                    }else if (element.equals("icon")){
                        Weather.current_image_url=parser.getAttributeValue(0);
                    Weather.current_image=GetURLBitmap(Weather.current_image_url);
                    }
                }

                if (element.equals("current_conditions") &&
                        eventCode==XmlPullParser.END_TAG){
                    break;
                }
            }
        }
        if (element !=null && element.equals("forecast_conditions")){
            while(true){
                int eventCode=parser.next();
                element=parser.getName();
                if (eventCode==XmlPullParser.START_TAG){
                    if (element.equals("day_of_week")){
                Weather.day[dayCounter].day_of_week=parser.
                getAttributeValue(0);
                    }else if (element.equals("low")){
                        Weather.day[dayCounter].low=parser.getAttributeValue(0);
                    }else if (element.equals("high")){
                        Weather.day[dayCounter].high=parser.getAttributeValue(0);
                    }else if (element.equals("icon")){
                    Weather.day[dayCounter].image_url=parser.getAttributeValue(0);
                    Weather.day[dayCounter].image=GetURLBitmap(Weather.day[dayCounter].image_
                    url);
                    }else if (element.equals("condition")){
                    Weather.day[dayCounter].condition=parser.getAttributeValue(0);
                    }
                }

                if (element.equals("forecast_conditions") &&
                        eventCode==XmlPullParser.END_TAG){
                    dayCounter++;
                    break;
                }
            }
```

```
            }
        }
        is.close();
    }
```

最后，在 AndroidManifest.xml 文件中注册 WeatherService，并声明连接互联网和发送 SMS 短信的两个用户许可。

```
<service android:name=".Service.WeatherService"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.SEND_SMS"/>
```

## 11.3.5 用户界面

在用户界面设计上，采用可以在多个分页上快速切换的 Tab 标签页。与以往介绍过的 Tab 标签页的使用方法不同，WeatherDemo 示例的 Tab 标签页将每个标签页与一个 Activity 关联在一起，这样做的好处就是可以将不同标签页的代码放在不同的文件中，而且每个标签页都可以有独立的选项菜单。

WeatherDemo 类是继承 TabActivity 的 Tab 标签页，共设置 3 个标签页。TAB1 标签页的标题为"天气预报"，关联的 Activity 为 WeatherActivity；TAB2 标签页的标题为"历史数据"，关联 Activity 为 HistoryActivity；TAB3 标签页的标题为"系统设置"，关联 Activity 为 SetupActivity。

WeatherDemo.java 文件的完整代码如下：

```
package edu.hrbeu.WeatherDemo;

import android.app.TabActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TabHost;

public class WeatherDemo extends TabActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TabHost tabHost=getTabHost();
        tabHost.addTab(tabHost.newTabSpec("TAB1")
        .setIndicator("天气预报", getResources().getDrawable(R.drawable.tab_weather))
         .setContent(new Intent(this, WeatherActivity.class)));

        tabHost.addTab(tabHost.newTabSpec("TAB2")
        .setIndicator("历史数据",getResources().getDrawable(R.drawable.tab_history))
         .setContent(new Intent(this, HistoryActivity.class)));
```

```

            tabHost.addTab(tabHost.newTabSpec("TAB3")
            .setIndicator("系统设置",getResources().getDrawable(R.drawable.tab_setup))
             .setContent(new Intent(this, SetupActivity.class)));
        }
    }
```

WeatherDemo.java 中的代码只是用户界面的框架，设置了 Tab 标签页的图标、标题和所关联的 Activity，标签页中的具体显示内容还要依赖于每个 Activity 所设置的界面布局。下面就分别介绍 WeatherActivity、HistoryActivity 和 SetupActivity。

### 1. WeatherActivity

WeatherActivity 主要用来显示天气信息，如图 11.4 所示。WeatherActivity 在启动时并不能够显示最新的天气信息，用户需要通过选项菜单的“启动服务”开启后台服务，然后单击“刷新”按钮获取最新的天气状况。此外，选项菜单还提供“停止服务”和“退出”选项。

WeatherActivity 使用的布局文件是 tab_weather.xml，这是个较为繁琐的界面布局，多次的使用了垂直和水平的线性布局。WeatherActivity 的界面布局和代码并不难理解，因此这里不再给出 WeatherActivity.java 和 tab_weather.xml 具体代码。

### 2. HistoryActivity

HistoryActivity 主要用来显示 SQLite 数据库中的短信服务信息，显示的内容包括发送者的手机号码、时间和回复短信内容，如图 11.5 所示。为了能够以列表的形式显示多行数据，并定制每行数据的布局，使用了以往章节没有介绍过的 ListActivity（Android.app.ListActivity）。

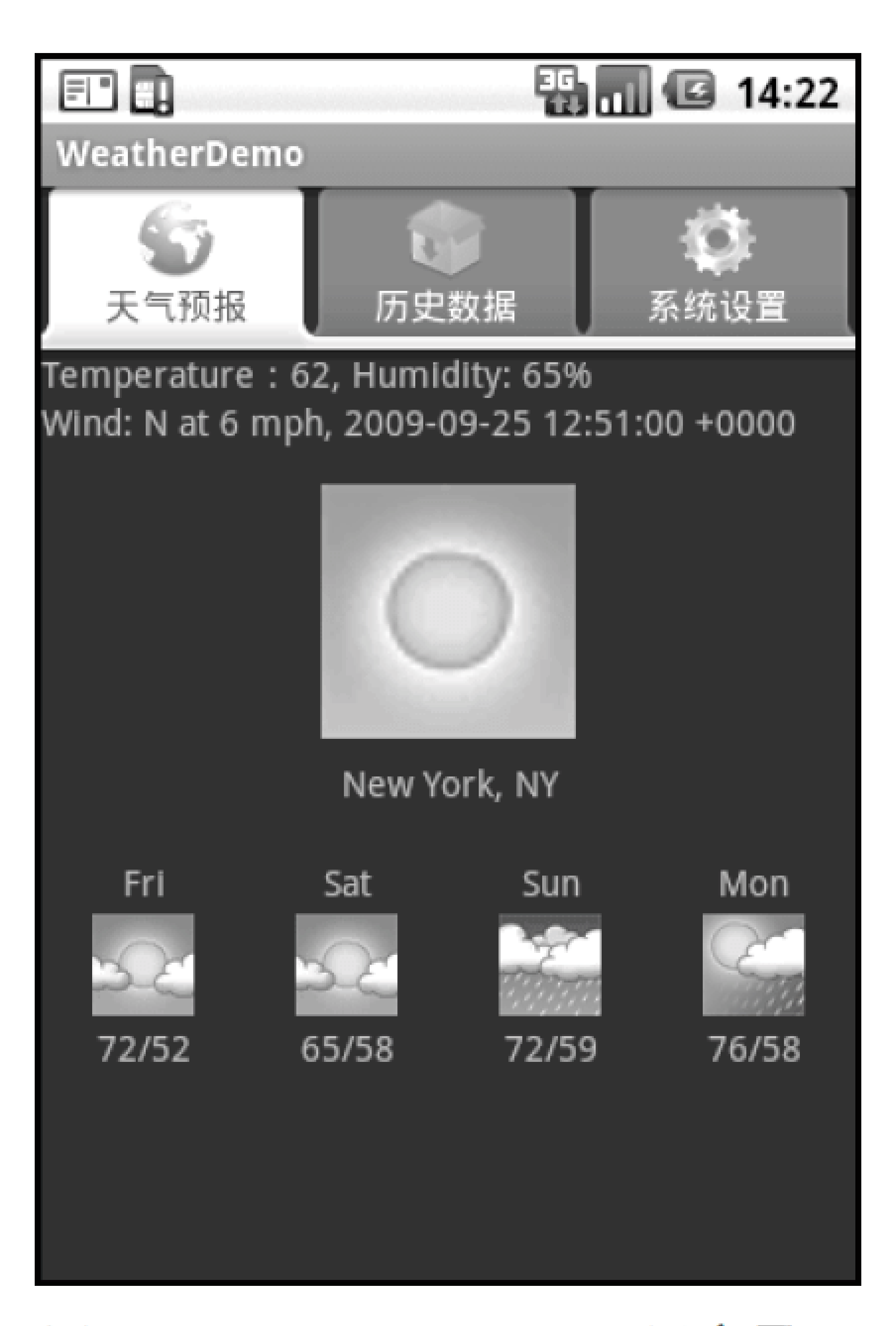


图 11.4　WeatherActivity 用户界面




**图 11.5　HistoryActivity 用户界面**

ListActivity 可以不通过 setContentView()设置布局，也不必重载 onCreate()函数，而直接将显示列表加载到 ListActivity，增加了使用的便利性。在 WeatherDemo 示例中，仍然使用 setContentView()设置布局，这样做的好处是可以在界面中设置更为复杂的显示元素，例如在列表上方增加了提示信息“SQLite 数据库中的短信服务信息”。下方的代码是 HistoryActivity.java 文件的 onCreate()函数中的设置布局和加载适配器的关键代码：

```
setContentView(R.layout.tab_history);
setListAdapter(dataAdapter);
```

tab_history.xml 是 HistoryActivity 的布局文件，下面先分析一下 tab_history.xml 的内容。tab_history.xml 文件的完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/black">

    <TextView  android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="SQLite 数据库中的短信服务信息：">
    </TextView>
    <ListView android:id="@android:id/list"
```

```
13            android:layout_width="fill_parent"
14            android:layout_height="wrap_content"
15            android:layout_marginTop="2dip">
16        </ListView>
17    </LinearLayout>
```

tab_history.xml 在代码的第 12 行至第 16 行增加了 ListView 控件，并使用系统的 ID 值“@android:id/list”，ListView 的数据适配器是通过 setListAdapter(dataAdapter) 设置的。ListView 使用的是自定义布局，布局保存在 data_row.xml 文件中，data_row.xml 的完整代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/white"
    android:layout_marginTop="2dip">

    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView android:id="@+id/data_row_01"
            android:layout_gravity="center_vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textSize="12dip"
            android:textColor="@drawable/black"/>

        <TextView android:id="@+id/data_row_02"
            android:layout_gravity="center_vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textSize="12dip"
            android:textColor="@drawable/black"
            android:layout_marginTop="3dip"/>
    </LinearLayout>
</LinearLayout>
```

Android 提供的数据适配器仅允许保存字符串数组或列表对象，如果希望使用自定义布局，则需要实现自定义的数据适配器，并继承 Android 提供的 BaseAdapter (Android.widget.BaseAdapter) 对象。自定义的数据适配器在 SmsAdapter.java 文件中，其完整代码如下：

```
package edu.hrbeu.WeatherDemo.SMS;
```

```java

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import edu.hrbeu.WeatherDemo.DB.DBAdapter;
import edu.hrbeu.WeatherDemo.R;


public class SmsAdapter extends BaseAdapter{
    private LayoutInflater mInflater;
    private static DBAdapter dbAdapter ;
    private static SimpleSms[] smsList ;

    public SmsAdapter(Context context){
        mInflater= LayoutInflater.from(context);
        dbAdapter= new DBAdapter(context);
        dbAdapter.open();
        smsList= dbAdapter.GetAllSms();
    }

    public static void RefreshData(){
        smsList= dbAdapter.GetAllSms();
    }
    @Override
    public int getCount(){
        if (smsList==null)
            return 0;
        else
            return smsList.length;
    }
    @Override
    public Object getItem(int position) {
        if (smsList==null)
            return 0;
        else
            return smsList[position];
    }
    @Override
    public long getItemId(int position) {
        return position;
    }
```

```
46
47          @Override
48          public View getView(int position, View convertView, ViewGroup parent) {
49              ViewHolder holder;
50              if(convertView==null){
51                  convertView=mInflater.inflate(R.layout.data_row, null);
52                  holder=new ViewHolder();
53                  holder.textRow01=(TextView) convertView.findViewById(R.id.data_row_01);
54                  holder.textRow02=(TextView) convertView.findViewById(R.id.data_row_02);
55                  convertView.setTag(holder);
56              }
57              else{
58                  holder=(ViewHolder) convertView.getTag();
59              }
60
61              if (smsList != null){
62                  String row01Msg="("+position+")"+"发送者："+smsList[position].Sender+","+
                smsList[position].ReceiveTime;
63                  holder.textRow01.setText(row01Msg);
64                  holder.textRow02.setText(smsList[position].ReturnResult);
65              }
66              return convertView;
67          }
68
69          private class ViewHolder{
70              TextView textRow01;
71              TextView textRow02;
72          }
73      }
```

继承 BaseAdapter 类，则首先要重载 4 个函数，包括 getCount()、getItem()、getItemId()和 getView()。LayoutInflater 是将 XML 文件中的布局映射为 View 对象的类，在代码第 14 行进行了声明，在代码第 51 行，将 data_row.xml 文件映射为 View 对象。代码第 70 行和第 71 行的内容，需要对应 data_row.xml 文件中的界面元素。

### 3. SetupActivity

SetupActivity 主要用来保存和恢复用户设置的运行参数，第一次启动或恢复默认设置(在选项菜单中)后，界面上会显示系统的默认设置，包括城市名称、更新频率、是否提供短信服务、是否记录短信服务数据信息和短信服务的关键字。SetupActivity 用户界面如图 11.6 所示。




SetupActivity.java 文件的主要功能集中在 RestoreDefaultSetup()、UpdateUI()和 SaveConfig()三

个函数上。RestoreDefaultSetup()用来恢复系统的默认配置;UpdateUI()会根据保存在Config类中的数据更新SetupActivity的界面控件;SaveConfig()根据界面配置更改Config类,然后调用数据库适配器的DBAdapter.SaveConfig()函数,将Config类中的配置数据写入数据库。

```
private void RestoreDefaultSetup(){
    Config.LoadDefaultConfig();
    UpdateUI();
    dbAdapter.SaveConfig();
}

private void UpdateUI(){
    cityNameView.setText(Config.CityName);
    refreshSpeedView.setText(Config.RefreshSpeed);
    smsServiceView.setChecked(Config.ProvideSmsService.equals("true")?
    true:false);
    saveSmsInfoView.setChecked(Config.SaveSmsInfo.equals("true")?true:false);
    keyWorkView.setText(Config.KeyWord);
}

private void SaveConfig(){
    Config.CityName= cityNameView.getText().toString().trim();
    Config.RefreshSpeed= refreshSpeedView.getText().toString();
    if (smsServiceView.isChecked()){
        Config.ProvideSmsService= "true";
    }else{
        Config.ProvideSmsService= "false";
    }
    if (saveSmsInfoView.isChecked()){
        Config.SaveSmsInfo= "true";
    }
    else{
        Config.SaveSmsInfo= "false";
    }
    Config.KeyWord= keyWorkView.getText().toString().trim();
    dbAdapter.SaveConfig();
}
```

最后,为了使定义的Activity和ListActivity生效,在AndroidManifest.xml文件中注册所有定义的组件。

```
<activity android:name=".WeatherDemo"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".WeatherActivity"/>
    <activity android:name=".HistoryActivity"/>
    <activity android:name=".SetupActivity"/>
```

# 附录 A




# Android 虚拟设备

Android 虚拟设备(AVD)能够通过 Android 命令行工具进行管理，包括 AVD 的创建、删除、移动和更新等。表 A.1 给出了 AVD 的管理命令及其参数说明。

**表 A.1　AVD 管理命令**

| 命　　令 | 参　　数 | 说　　明 | 备　　注 |
|---|---|---|---|
| android list avds | | 显示所有已知的 AVD，内容包括 AVD 的名称、路径和外观等 | |
| android create avd | -n <name> | AVD 名称 | 建立 AVD 的必备参数 |
| | -t <targetID> | Android 系统映像 ID | 使用 android list targets 命令获取 Android 系统映像的 ID 列表 |
| | -c <path>或<br>-c <size>[K\|M] | SD 卡映像文件路径或 SD 卡映像的容量 | 示例 1：-c path/to/sdcard<br>示例 2：-c 1000M |
| | -f | 强制建立 AVD | 如果新建立 AVD 与已有 AVD 的名称相同，则 Android 工具将提示“AVD 已经存在”，自动停止 AVD 的创建过程。使用该参数，Android 工具将自动删除已有同名 AVD，并建立新的 AVD |
| | -p <path> | 保存 AVD 文件和目录的位置 | |
| | -s <name>或<br>-s<width><br>-<height> | 指定 AVD 的外观，利用外观名称或长宽进行选择 | 示例 1：-s HVGA-L<br>示例 2：-s 320x240 |
| android delete avd | -n <name> | AVD 名称 | 删除 AVD 的必备参数 |
| android move avd | -n <name> | AVD 名称 | 移动 AVD 的必备参数 |
| | -p <path> | 移动后的 AVD 位置 | |
| | -r <new-name> | 重命名 AVD 的新名称 | |
| android update avds | | 为所有 Android 系统映像计算路径 | |

使用标准的 Android 系统映像创建 AVD 时，Android 工具允许用户选择虚拟设备所支持的硬件列表，表 A. 2 列举出可以选择的硬件及其默认值。同时，用户也可以在 AVD 的 config. ini 文件中找到相关的硬件选择设置。

表 A. 2 虚拟设备硬件令列表

| 特征 | 描述 | 属性 |
|---|---|---|
| RAM 容量 | 物理 RAM 容量，单位为兆，默认值为 96 | hw. ramSize |
| 触摸屏 | 设备是否支持触摸屏，默认支持 | hw. touchScreen |
| 轨迹球 | 设备是否支持轨迹球，默认支持 | hw. trackBall |
| QWERTY 键盘 | 设备是否支持 QWERTY 键盘，默认支持 | hw. keyboard |
| DPad 键 | 设备是否支持 DPad 键，默认支持 | hw. dPad |
| GSM 调制解调器 | 设备是否支持 GSM 调制解调器，默认支持 | hw. gsmModem |
| 摄像头 | 设备是否支持摄像头，默认不支持 | hw. camera |
| 摄像头水平方向像素的最大值 | 默认值为 640 | hw. camera. maxHorizontalPixels |
| 摄像头垂直方向像素的最大值 | 默认值为 480 | hw. camera. maxVerticalPixels |
| GPS | 设备是否支持 GPS，默认支持 | hw. gps |
| 电池 | 设备是否支持电池，默认支持 | hw. battery |
| 加速度计 | 设备是否支加速度计设备，默认支持 | hw. accelerometer |
| 录音 | 设备是否支录音，默认支持 | hw. audioInput |
| 声音回放 | 设备是否支持声音回放，默认支持 | hw. audioOutput |
| SD 卡 | 设备是否支持虚拟的 SD 卡，默认支持 | hw. sdCard |
| 缓存分区 | 是否在设备中使用缓存分区，默认使用 | disk. cachePartition |
| 缓存分区容量 | 缺省值为 66MB | disk. cachePartition. size |

# 附录 B appendix B

# Android API 简介

| 包 名 称 | 说 明 |
|---|---|
| android | 包含了标准 Android 应用使用的资源类 |
| android. app | 封装了全部 Android 应用模型的高级类 |
| android. appwidget | Android 允许应用程序发布嵌入到其他应用程序中的视图。这些视图被称为窗口部件，由 AppWidget providers 发布。可以包含窗口部件的部件被称为 AppWidget host |
| android. content | 包含了用来在设备上访问和发布数据的类 |
| android. content. pm | 包含了用来访问应用程序软件包信息的类，其中包括它的行为、权限、服务、签名和供应商等信息 |
| android. content. res | 包含了用来访问应用程序资源的类，比如原始的有价值的文件、颜色、可绘区、多媒体或者包中的其他文件，还有影响到应用程序如何表现的重要的设备配置信息(如位置、输入类型等) |
| android. database | 包含用来探究通过内容提供商返回的数据的类 |
| android. database. sqlite | 包含了 SQLite 数据库管理类，应用程序可以用它来管理自己的私有数据库 |
| android. graphics | 提供了低级别的图形工具，比如画布、颜色过滤器、点和矩形，可以对屏幕绘图进行直接控制 |
| android. graphics. drawable | 提供了显示的多种视觉元素的类，比如位图和斜率 |
| android. graphics. drawable. shapes | 包含了用来绘制几何图形的类 |
| android. hardware | 提供对可能并不是每台 Android 设备都具有的硬件设备的支持 |
| android. inputmethodservice | 写入方法的基础类 |
| android. location | 定义了 Android 基于位置服务的类 |
| android. media | 提供了管理语音和视频中的多种媒体接口的类 |
| android. net | 协助进行网络存取的类，包括标准 java. net. * 应用程序接口 |
| android. net. http | |
| android. net. wifi | 提供了管理设备上 Wi-Fi 功能的类 |

续表

| 包 名 称 | 说 明 |
|---|---|
| android. opengl | 提供 OpenGL 功能 |
| android. os | 提供设备上的基本操作系统服务,消息路由和进程间通信 |
| android. preference | 提供管理应用程序性能和执行 UI 参数选择的类 |
| android. provider | 提供了方便的类用来访问 Android 的内容提供商 |
| android. sax | 简化 SAX 操作的框架 |
| android. speech | |
| android. telephony | 提供了应用程序接口用来监测基本电话信息,比如网络类型和连接状态,还有控制电话号码的功能 |
| android. telephony. gsm | 提供了应用程序接口用来使用 GSM 特有的电话性能,比如文本/数据/PDU SMS 消息 |
| android. test | 用来书写 Android 测试框架 |
| android. test. mock | 提供了存根或者多种 Android 框架基础模拟的功能类 |
| android. test. suitebuilder | 支持试运行类的功能类 |
| android. text | 提供了用来显示或追踪屏幕上文本和文本间距的类 |
| android. text. format | |
| android. text. method | 提供了用来监测或修改键盘输入的类 |
| android. text. style | 提供了用来显示或改变文本风格的类 |
| android. text. util | 把确定的文本转化为可点击的链接并创建 RFC 822 消息类型(SMTP)标记 |
| android. util | 提供了一般的功能方法,比如日期/时间控制、64 位编解码器、字符串和数字转换方法和 XML 功能 |
| android. view | 提供用来显示控制屏幕布局和用户交互的基本用户接口类 |
| android. view. animation | 提供用来控制中间动画列表的类 |
| android. view. inputmethod | 用来进行视图和输入方法(比如软键盘)交互的框架类 |
| android. webkit | 提供了浏览互联网的工具 |
| android. widget | 界面组件,包含了应用程序屏幕上使用的 UI 元素 |
| com. android. internal. os | |
| dalvik. bytecode | 提供了围绕 Dalvik 字节码的类 |
| dalvik. system | 提供了 Dalvik VM 特有的功能和系统信息的类 |
| java. awt. font | |
| java. beans | |
| java. io | 提供了流输入输出方法,文件系统访问和串行化 |
| java. lang | 提供了 Android 环境的核心类 |

续表

| 包　名　称 | 说　　明 |
| --- | --- |
| java. lang. annotation | 定义了注释支持必要的接口和例外 |
| java. lang. ref | |
| java. lang. reflect | |
| java. math | 提供了高精度的整数和小数 |
| java. net | 提供了网络相关功能，比如流和数据包套接口，互联网地址操作和处理 HTTP 请求 |
| java. nio | 提供了协助控制数据的缓冲区 |
| java. nio. channels | 提供了连接数据资源比如文件、套接口或其他允许输入输出数据的结构的方法 |
| java. nio. channels. spi | nio channels 的服务提供类 |
| java. nio. charset | 提供了字节和不同字符集之间的转换服务 |
| java. nio. charset. spi | nio charset 的服务提供类 |
| java. security | 提供了所有组成 Java 安全框架的类和接口 |
| java. security. acl | 提供了用来建立访问控制列表的类和接口 |
| java. security. cert | 提供了用来生成、管理和证实 X. 509 证书的所有类和接口 |
| java. security. interfaces | 提供安全密钥的接口，包括：(1)使用 PKCS＃1 标准的 RSA 非均匀加密算法的密钥；(2)以 FIPS-186 表示的数字签名算法(DSA)的密钥；(3)一般椭圆曲线不均匀算法的密钥 |
| java. security. spec | 为加密和签名算法的特别的密钥和参数提供了类和接口 |
| java. sql | 提供了标准接口用来访问基于 SQL 的数据库 |
| java. text | java. text 包允许打开程序中来自自然语言的文本 |
| java. util | 提供了功能类的扩展 |
| java. util. concurrent | 功能类一般在同时运行的程序中比较常用 |
| java. util. concurrent. atomic | 支持在单个变量上解除锁定的线程安全编程工具包 |
| java. util. concurrent. locks | 接口和类提供了一个框架用来锁住并等待来自于内置同步和检测器的唯一条件 |
| java. util. jar | java. jar 包提供了读取和写入 Java 文档文件或 JAR 文件的权限 |
| java. util. logging | 允许为任何应用程序添加记录 |
| java. util. prefs | 提供了参数选择机制，这是一种把配置数据(键值对)写到持续的数据存储中并进行检索的方法 |
| java. util. regex | 提供了正则表达式的一种执行，这对基于形式的字符串匹配、查找以及替换来说是有用的 |
| java. util. zip | 包含了压缩和解压缩 ZIP 和 GZIP 格式文件的类 |

续表

| 包　名　称 | 说　　明 |
| --- | --- |
| javax. crypto | 为加密算法应用程序提供类和接口用来执行加密、解密和关键协议等算法 |
| javax. crypto. interfaces | 这个包提供了用来执行 Diffie-Hellman （DH）关键协议算法像 PKCS＃3 的接口 |
| javax. crypto. spec | 提供了加密需要的指定密钥和参数的类和接口 |
| javax. microedition. khronos. egl | |
| javax. microedition. khronos. opengles | |
| javax. net | 提供了工厂类用来创建套接字和服务套接字 |
| javax. net. ssl | 提供了用来执行和编写基于 SSL 协议 SSSLv3. 0 或 TLSv1. 2 的安全套接字的所有类和接口 |
| javax. security. auth | 提供了用来执行和编写用户授权的不同方法和基于用户授权的角色的类和接口 |
| javax. security. auth. callback | 提供了用来与应用程序进行交互以执行用户授权和认可的进程的类和接口 |
| javax. security. auth. login | 这个包提供了一个基于 UNIX-PAM 模型的概念的便携可折叠的认证系统 |
| javax. security. auth. x500 | 提供了用来存储 X. 500 代理以及其认证的类 |
| javax. security. cert | 只是由于兼容性才被使用 |
| javax. sql | 提供了用来访问基于 SQL 的数据库的标准接口 |
| javax. xml | 提供了一个带有有用的 XML 常量的公共类 |
| javax. xml. parsers | 提供了用来解析 XML 文档的条件并且建立文档对象模型树 |
| junit. framework | junit 测试框架 |
| junit. runner | 支持 junit 测试框架的公共类 |
| org. apache. http | HTTP 协议的核心接口和类 |
| org. apache. http. auth | 对于服务器来说的客户端 HTTP 认证的 API |
| org. apache. http. auth. params | 用来配置 HttpAuth 的参数 |
| org. apache. http. client | 用来进行客户端 HTTP 通信的 API 和 HttpClient 模型的入口点 |
| org. apache. http. client. entity | |
| org. apache. http. client. methods | 请求执行各种 HTTP 方法，比如 GET 和 POST |
| org. apache. http. client. params | 用来配置 HttpClient 的参数 |
| org. apache. http. client. protocol | 附加请求及回应拦截 |

续表

| 包 名 称 | 说 明 |
|---|---|
| org. apache. http. client. utils | HttpClient 的助手和公共类 |
| org. apache. http. conn | 客户端连接管理和控制 HttpConn 中心的 API |
| org. apache. http. conn. params | 用来配置 HttpConn 的参数 |
| org. apache. http. conn. routing | 客户端路由代理和跟踪 API,HttpConn 的一部分 |
| org. apache. http. conn. scheme | |
| org. apache. http. conn. ssl | TLS/SSL HttpConn API 特有的部分 |
| org. apache. http. conn. util | |
| org. apache. http. cookie | 通过 cookies 进行客户端状态管理的 API |
| org. apache. http. cookie. params | 用来配置 HttpCookie 的参数 |
| org. apache. http. entity | HTTP 消息实体的代理 |
| org. apache. http. impl | org. apache. http 中接口的默认执行 |
| org. apache. http. impl. auth | |
| org. apache. http. impl. client | |
| org. apache. http. impl. conn | |
| org. apache. http. impl. conn. tsccm | 一个线程安全的客户端连接管理的一次执行 |
| org. apache. http. impl. cookie | |
| org. apache. http. impl. entity | org. apache. http. entity 中接口的默认执行 |
| org. apache. http. impl. io | org. apache. http. io 中接口的默认执行 |
| org. apache. http. io | HTTP 部分的传输层 |
| org. apache. http. message | HTTP 消息执行的选择 |
| org. apache. http. params | HTTP 部分的参数框架 |
| org. apache. http. protocol | HTTP 协议执行框架 |
| org. apache. http. util | 出于各种目的的带有静态辅助方法的公共类 |
| org. json | |
| org. w3c. dom | 提供了正式的 W3C 的 Java 绑定的文档对象模型,2 级核心 |
| org. xml. sax | 提供了核心 SAX APIs |
| org. xml. sax. ext | SAX2 条件的接口,保证了 SAX 驱动并不一定要支持 |
| org. xml. sax. helpers | 辅助方法类 |
| org. xmlpull. v1 | |
| org. xmlpull. v1. sax2 | |

# 附录 C




# ADB 命 令

| 类　别 | 命　令 | 说　明 |
|---|---|---|
| 参数 | －d | 指定 adb 命令发往连接的 USB 设备(备注：如果连接两个或两个以上的 USB 设备，则返回错误提示) |
| | －e | 指定 adb 命令仅发往运行的 Android 模拟器(备注：如果有两个或两个以上的 Android 模拟器在运行，则返回错误提示) |
| | －s ＜serialNumber＞ | 指定 adb 命名链接的模拟器或设备，模拟器或设备通过“emulator-端口”的方式指定，例如 emulator-5556 |
| 通用命令 | devices | 显示所有链接的模拟器或设备列表 |
| | help | 显示 adb 支持的命令列表 |
| | version | 显示 adb 的版本号 |
| 调试命令 | logcat[＜option＞]<br>[＜filter-specs＞] | 显示日志数据 |
| | bugreport | 显示 dumpsys、dumpstate 和 logcat 数据 |
| | jdwp | 显示指定设备的 JDWP 进程列表 |
| 数据命令 | install ＜path-to-apk＞ | 将 Android 应用程序安装到模拟器或设备 |
| | pull ＜remote＞ ＜local＞ | 将模拟器或设备上的指定文件复制到开发主机 |
| | push ＜local＞ ＜remote＞ | 将开发主机上的指定文件复制到模拟器或设备 |
| 网络与端口命令 | forward ＜local＞ ＜remote＞ | 将本地端口的 Socket 链接的数据转发到特定的模拟器或设备端口 |
| | ppp ＜tty＞ [parm] | 在 USB 上运行 PPP |
| 脚本命令 | get-serialno | 显示 adb 链接的模拟器或设备的序列号 |
| | get-state | 显示 adb 链接的模拟器或设备的状态 |
| | wait-for-device | 阻止程序运行，等待设备就绪 |

续表

| 类　　别 | 命　　令 | 说　　明 |
| --- | --- | --- |
| 服务器命令 | start-server | 启动 adb 服务进程 |
| | kill-server | 终止 adb 服务进程 |
| Shell 命令 | shell | 针对目标模拟器或设备启动一个 Shell |
| | shell [<shellCommand>] | 针对目标模拟器或设备启动一个 Shell,并直接执行 Shell 命令<shellCommand> |




# 附录 D

# AndroidManifest 文件

每一个 Android 应用程序都有一个 AndroidManifest.xml 文件，用以在程序运行前向 Android 系统声明程序的相关信息，这些信息包括应用程序需要的许可、程序运行的最低 SDK 版本、程序运行所需要的硬件支持和函数库等等。

AndroidManifest.xml 文件主要由元素、属性和类声明等重要部分组成。在 AndroidManifest.xml 中，多数元素的出现次数没有限制，而且在同一层次的元素没有顺序要求，例如<activity>、<provider>和<service>元素可以任何顺序排列，但<manifest>和<application>元素只能在 AndroidManifest.xml 中出现一次，而且必须出现一次。AndroidManifest.xml 文件的元素与属性关系参考表 D.1，通用结构如下所示：

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter>… </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter>… </intent-filter>
            <meta-data/>
        </service>
```

```
            <receiver>
                <intent-filter>… </intent-filter>
                <meta-data />
            </receiver>
            <provider>
                <grant-uri-permission />
                <meta-data />
            </provider>
            <uses-library />
            <uses-configuration />
        </application>
    </manifest>
```

在通常情况下，元素的属性是用来描述如何解释元素内容，一般都是可以省略的。但在AndroidManifest.xml中，为了使元素具有意义，一些特定的属性是不能够省略的，例如<manifest>元素的<xmlns:android>属性和< package >属性。如果需要为同一个元素的属性指定多个值，则需要重复书写这个元素，书写方法如下所示。

```
<intent-filter …>
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    …
</intent-filter>
```

许多元素的定义需要声明类名，例如<activity>、<service>和<receiver>等元素。如果在直接通过android:name属性声明类名，则声明的类名必须包含完成的包名称。但如果在<manifest>元素中使用了<package>属性，则可以在android:name属性声明类名时不包含完整的包名称。区别可以参考下面的两段代码。

完整类名：

```
<manifest …>
    <application …>
        <service android:name="edu.hrbeu.MyService" …>
            …
        </service>
        …
    </application>
</manifest>
```

简化类名：

```
<manifest package="edu.hrbeu" …>
    <application … >
        <service android:name=".MyService" …>
            …
        </service>
        …
    </application>
</manifest>
```

表 D.1　AndroidManifest 中元素及其属性

| 元素 | 父元素 | 子元素 | 可用属性 | 描述 |
|---|---|---|---|---|
| <action> | <intent-filter> | | android:name | 用以定义<intent-filter>元素的过滤参数 |
| <activity> | <application> | <intent-filter><br><meta-data> | allowTaskReparenting<br>android:alwaysRetainTaskState<br>android:clearTaskOnLaunch<br>android:configChanges<br>android:enabled<br>android:excludeFromRecents<br>android:exported<br>android:finishOnTaskLaunch<br>android:icon<br>android:label<br>android:launchMode<br>android:multiprocess<br>android:name<br>android:noHistory<br>android:permission<br>android:process<br>android:screenOrientation<br>android:stateNotNeeded<br>android:taskAffinity<br>android:theme<br>android:windowSoftInputMode | 声明 Activity，每个 Activity 子类必须声明一个<activity>元素，否则 Activity 子类无法运行 |

续表

| 元素 | 父元素 | 子元素 | 可用属性 | 描述 |
| --- | --- | --- | --- | --- |
| <activity-alias> | <application> | <intent-filter><br><meta-data> | android:enabled<br>android:exported<br>android:icon<br>android:label<br>android:name<br>android:permission<br>android:targetActivity | 声明 Activity 的别名，使用 android:targetActivity 属性定义原始Activity |
| <application> | <manifest> | <activity><br><activity-alias><br><service><br><receiver><br><provider><br><uses-library> | android:allowClearUserData<br>android:allowTaskReparenting<br>android:debuggable<br>android:description<br>android:enabled<br>android:hasCode<br>android:icon<br>android:label<br>android:manageSpaceActivity<br>android:name<br>android:permission<br>android:persistent<br>android:process<br>android:taskAffinity<br>android:theme | Android 应用程序声明，子元素包含应用程序所有的组成模块，所定义的属性将影响所有模块 |
| <category> | <intent-filter> | | android:name | 用以定义<intent-filter>元素的过滤参数 |

续表

| 元　　素 | 父　元　素 | 子　元　素 | 可 用 属 性 | 描　　述 |
|---|---|---|---|---|
| <data> | <intent-filter> | | android:host<br>android:mimeType<br>android:path<br>android:pathPattern<br>android:pathPrefix<br>android:port<br>android:scheme | 用以定义<intent-filter>元素参数，可以是数据类型、URI 或数据类型+URI |
| <grant-uri-permission> | <provider> | | android:path<br>android:pathPattern<br>android:pathPrefix | 定义内容提供者支持的数据子集 |
| <instrumentation> | <manifest> | | android:functionalTest<br>android:handleProfiling<br>android:icon<br>android:label<br>android:name<br>android:targetPackage | 声明 Instrumentation 类，用于监控应用程序与 Android 系统的交互过程 |
| <intent-filter > | <activity><br><activity-alias><br><service><br><receiver> | <action>(必有属性)<br><category><br><data> | android:icon<br>android:label<br>android:priority | 为 Activity、服务和广播消息接收器定义所需要接收的 Intent 类型，过滤参数一般由子元素< action >、<category>和<data>进行定义 |

续表

| 元　素 | 父元素 | 子元素 | 可用属性 | 描　述 |
|---|---|---|---|---|
| <manifest > | | <application><br>（必有属性）<br><instrumentation><br><permission><br><permission-group><br><permission-tree><br><uses-configuration><br><uses-permission><br><uses-sdk> | xmlns:android<br>package<br>android:sharedUserId<br>android:sharedUserLabel<br>android:versionCode<br>android:versionName | The root element of the file. It must contain an element and specify attributes.<br>AndroidManifest. xml 文件的根元素，必须声明<application>子元素、xlmns:android 属性和 package 属性 |
| <meta-data > | <activity><br><activity-alias><br><service><br><receiver> | | android:name<br>android:resource<br>android:value | 定义一个名称/值对，数据类型可以使用所有父元素所支持的数据类型。每个父元素可以定义多个<meta-data>子元素 |
| <permission> | <manifest> | | android:description<br>android:icon<br>android:label<br>android:name<br>android:permissionGroup<br>android:protectionLevel | 定义一个安全许可，可以控制应用程序访问受到限制的模块和特性 |
| <permission-group > | <manifest> | | android:description<br>android:icon<br>android:label<br>android:name | 为一组相关的许可定义一个许可组，每个独立的许可通过<permission>元素中 permissionGroup 属性加入许可组 |
| <permission-tree > | <manifest> | | android:icon<br>android:label<br>android:name | 声明许可树的前缀名称，应用程序能够通过 PackageManager. addPermission(). Names，动态地向许可树中添加新的许可 |

续表

| 元素 | 父元素 | 子元素 | 可用属性 | 描述 |
| --- | --- | --- | --- | --- |
| <provider> | <application> | <meta-data><br><grant-uri-permission> | android:authorities<br>android:enabled<br>android:exported<br>android:grantUriPermissions<br>android:icon<br>android:initOrder<br>android:label<br>android:multiprocess<br>android:name<br>android:permission<br>android:process<br>android:readPermission<br>android:syncable<br>android:writePermission | 声明内容提供者，每个内容提供者子类必须声明一个<provider>元素，否则 Android 系统无法找到内容提供者 |
| <receiver> | <application> | <intent-filer><br><meta-data> | android:enabled<br>android:exported<br>android:icon<br>android:label<br>android:name<br>android:permission<br>android:process | 声明广播消息接收器，每个广播消息接收器子类必须声明一个<receiver>元素。广播消息接收者能够接收 Android 系统或其他应用程序发出的广播消息 |
| <service> | <application> | <intent-filer><br><meta-data> | android:enabled<br>android:exported<br>android:icon<br>android:label<br>android:name<br>android:permission<br>android:process | 声明服务，每个服务的子类必须声明一个<service>元素。服务是长时间运行的后台程序，或是供其他应用程序调用的通信 API |

续表

| 元素 | 父元素 | 子元素 | 可用属性 | 描述 |
| --- | --- | --- | --- | --- |
| <uses-configuration> | <manifest> | | android:reqFiveWayNav<br>android:reqHardKeyboard<br>android:reqKeyboardType<br>android:reqNavigation<br>android:reqTouchScreen | 声明应用程序运行需要的硬件和软件条件 |
| <uses-library> | <application> | | android:name | 声明应用程序必须链接的共享函数库，<uses-library>元素通知Android系统链接用户指定的函数库，例如maps或awt等 |
| <uses-permission> | <manifest> | | android:name | 声明应用程序所需要的许可 |
| <uses-sdk> | <manifest> | | android:minSdkVersion | 声明应用程序对不同版本Android系统的兼容性 |

# 读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602 室 计算机与信息分社营销室 收
邮编：100084　　　　电子邮件：jsjjc@tup.tsinghua.edu.cn
电话：010-62770175-4608/4409　　　　邮购电话：010-62786544

教材名称：Android 应用程序开发

ISBN：978-7-302-21699-5

**个人资料**

姓名：__________ 年龄：______ 所在院校/专业：______________

文化程度：________ 通信地址：__________________________

联系电话：________ 电子信箱：__________________________

**您使用本书是作为：**□指定教材 □选用教材 □辅导教材 □自学教材

**您对本书封面设计的满意度：**

□很满意 □满意 □一般 □不满意　改进建议____________________

**您对本书印刷质量的满意度：**

□很满意 □满意 □一般 □不满意　改进建议____________________

**您对本书的总体满意度：**

从语言质量角度看 □很满意 □满意 □一般 □不满意

从科技含量角度看 □很满意 □满意 □一般 □不满意

**本书最令您满意的是：**

□指导明确 □内容充实 □讲解详尽 □实例丰富

**您认为本书在哪些地方应进行修改？（可附页）**

______________________________________________

______________________________________________

**您希望本书在哪些方面进行改进？（可附页）**

______________________________________________

______________________________________________

# 电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。